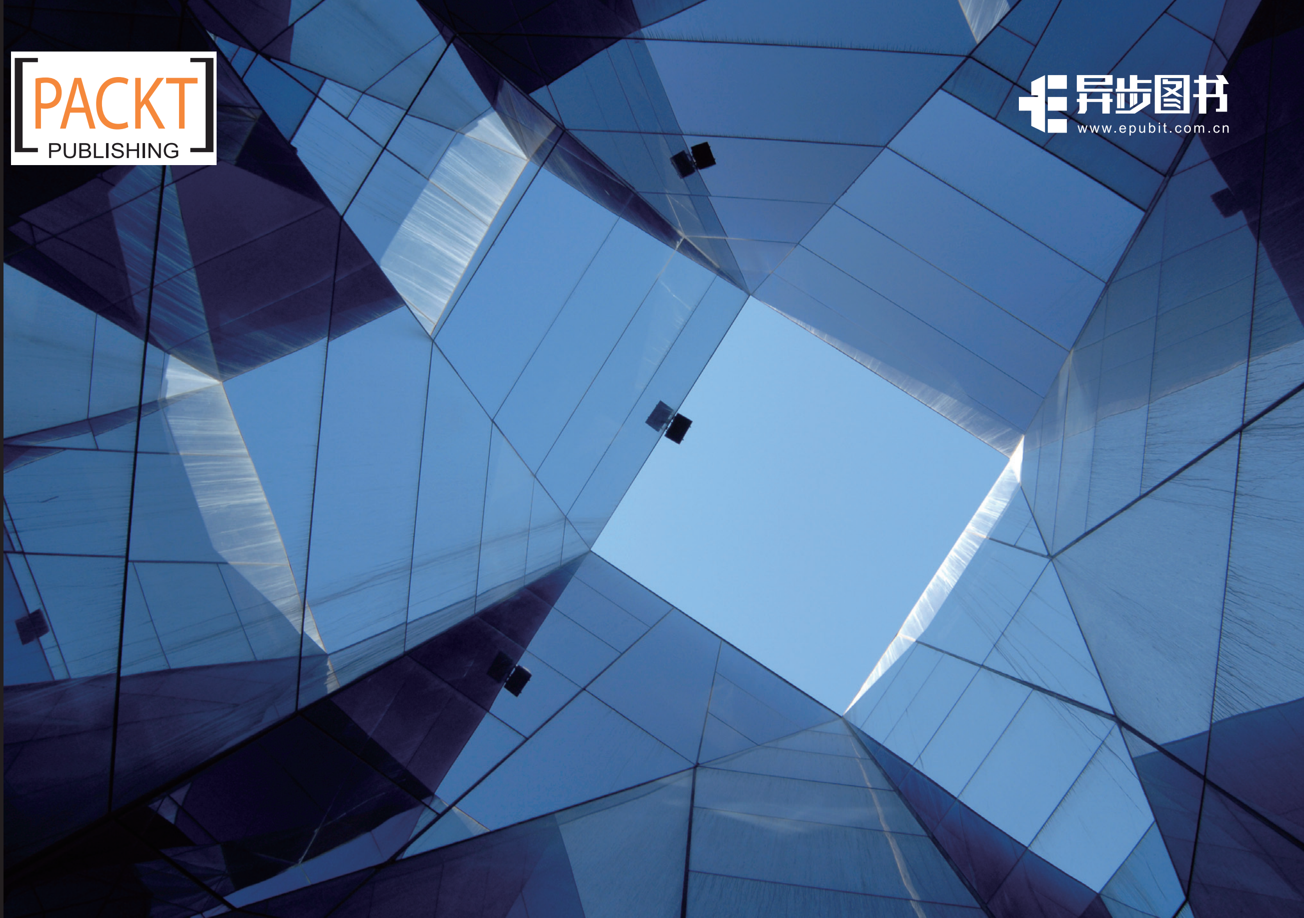


PACKT PUBLISHING

异步图书 www.epubit.com.cn

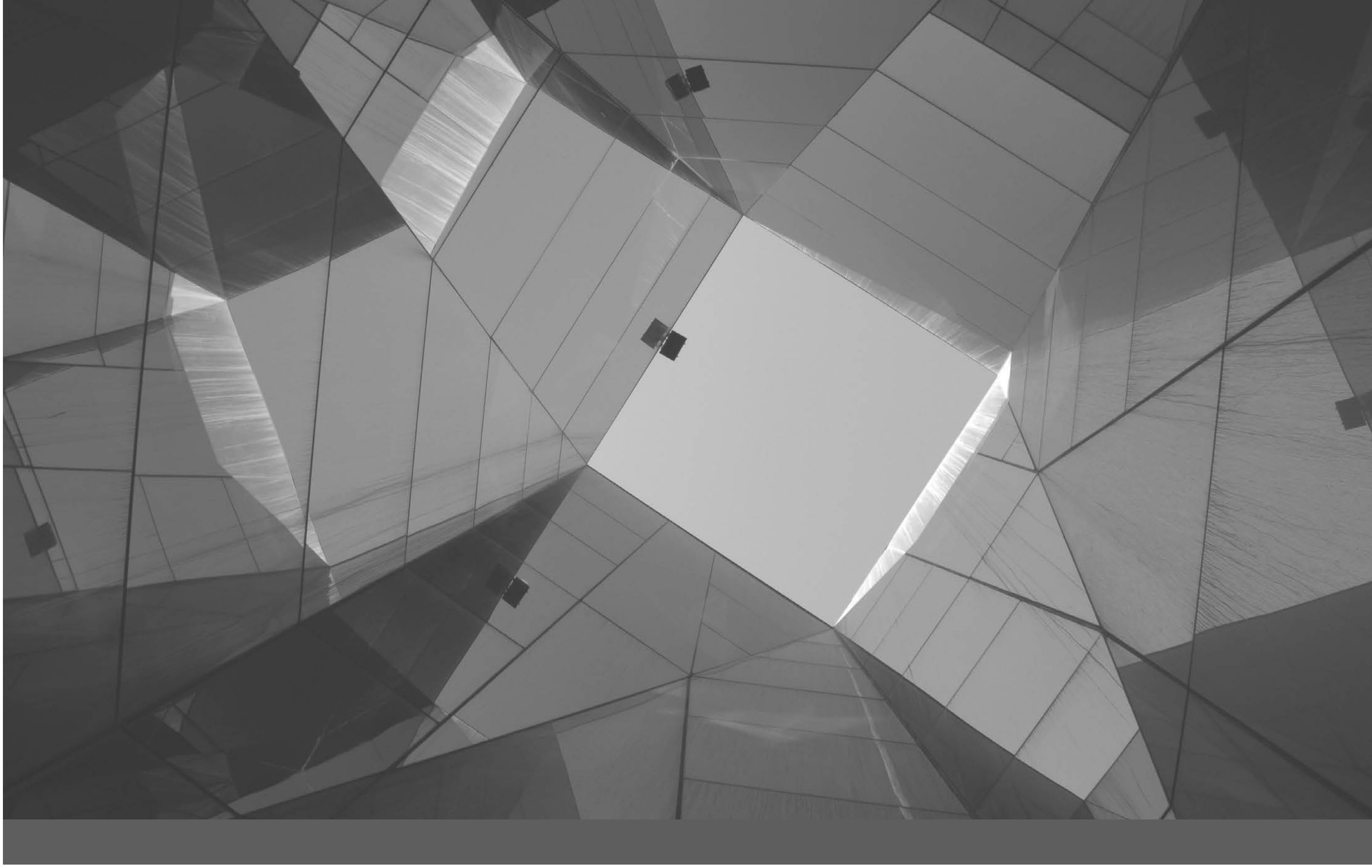
NLTK 基础教程

用NLTK和Python库构建机器学习应用

NLTK Essentials

[印度] Nitin Hardeniya 著
凌杰 译

中国工信出版集团 人民邮电出版社 POSTS & TELECOM PRESS



NLTK基础教程

用NLTK和Python库构建机器学习应用

[印度] Nitin Hardeniya 著
凌杰 译

人 民 邮 电 出 版 社
北 京

图书在版编目（C I P）数据

NLTK基础教程 ： 用NLTK和Python库构建机器学习应
用 / （印） 哈登尼亚 （Nitin Hardeniya） 著 ； 凌杰译
. -- 北京 ： 人民邮电出版社，2017.6
ISBN 978-7-115-45257-3

Ⅰ. ①N… Ⅱ. ①哈… ②凌… Ⅲ. ①软件工具－程序
设计 Ⅳ. ①TP311.561

中国版本图书馆CIP数据核字(2017)第079700号

版权声明

内容提要

NLTK 库是当前自然语言处理（NLP）领域最为流行、使用最为广泛的库之一，同时 Python 语言也已逐渐成为主流的编程语言之一。

本书主要介绍如何通过 NLTK 库与一些 Python 库的结合从而实现复杂的 NLP 任务和机器学习应用。全书共分为 10 章。第 1 章对 NLP 进行了简单介绍。第 2 章、第 3 章和第 4 章主要介绍一些通用的预处理技术、专属于 NLP 领域的预处理技术以及命名实体识别技术等。第 5 章之后的内容侧重于介绍如何构建一些 NLP 应用，涉及文本分类、数据科学和数据处理、社交媒体挖掘和大规模文本挖掘等方面。

本书适合 NLP 和机器学习领域的爱好者、对文本处理感兴趣的读者、想要快速学习 NLTK 的资深 Python 程序员以及机器学习领域的研究人员阅读。

◆ 著　[印度] Nitin Hardeniya
译　凌　杰
责任编辑　陈冀康
执行编辑　武晓燕
责任印制　焦志炜

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路 11 号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京鑫正大印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：10.75
字数：210 千字　2017 年 6 月第 1 版
印数：1 – 3 000 册　2017 年 6 月北京第 1 次印刷

著作权合同登记号　图字：01-2015-8290 号

定价：49.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广字第 8052 号

作者简介

Nitin Hardeniya 数据科学家，拥有 4 年以上从业经验，期间分别任职于 Fidelity、Groupon 和[24]7 等公司，其业务横跨各个不同的领域。此外，他还拥有 IIIT-H 的计算语言学硕士学位，并且是 5 项客户体验专利的作者。

他热衷于研究语言处理及大型非结构化数据，至少拥有 5 年日常使用 Python 的工作经验。他相信，用 Python 可以构建出大部分与数据科学相关问题的单点解决方案。

他将自己写这本书的经历看成是自己职业生涯的众多荣誉之一，希望用一种非常简单的形式为人们介绍与 NLP 和机器学习相关的、所有的这些复杂工具。在这本书中，他为读者提供了一种变通方法，即使用一些相关特定能力的 Python 库，如 NLTK、scikit-learn、panda 和 NumPy 等。

审阅者简介

Afroz Hussain 数据科学家，目前在 PredictifyMe 公司从事与美国基础数据科学、机器学习起步相关的研究。他在数据科学领域拥有丰富的项目经验、多年使用 Python、scikit-learn，以及基于 NLTK 进行文本挖掘的工作经历。他拥有 10 年以上的编程经验以及与数据分析和商业智能项目相关的软件开发经验。此外，他还通过在线课程以及参加 Kaggle 比赛等活动，获得了不少数据科学领域的新技能。

Sujit Pal 目前就职于 Elsevier 实验室，这是一个包含了 Reed-Elsevier PLC 工作组在内的研发团队。他的兴趣主要集中在信息检索、分布式处理、本体开发、自然语言处理和机器学习这几个领域。而且，他也很喜欢用 Python、Scala 和 Java 来编写自己的代码。他充分整合了自己在这些方面的技能，帮助公司改进了不同产品的一些特性并构建了一些新特性。他深信自己需要终身学习，并且也在博客：sujitpal.blogspot.com 中分享其经验。

Kumar Raj 第二代数据科学家，目前就职于惠普软件的研发部门，为其提供相关的解决方案。在那里，他主要负责开发以惠普软件产品为核心的分析层。他毕业于印度理工学院 Kharagpur 技术分校，并具有两年以上各种大数据分析领域的工作经验，涉及文本分析、网页抓取及检索、人力资源分析、虚拟系统的性能优化，以及气候变化的预测等。

译者序

说来也凑巧，在我签下这本书的翻译合同时，这个世界好像还不知道 AlphaGo 的存在。而在我完成这本书的翻译之时，Master 已经对人类顶级高手连胜 60 局了。至少从媒体的热度来看，的确在近几年，人工智能似乎是越来越火了。其原因是 Google 在汽车驾驶和围棋这两个领域的项目得到了很好的进展和宣传，而这两个领域在过去被很多人想当然地认为是人类的专属领域。因此在专属领域接连被突破情况下，一些人得了“机器恐惧症”。例如高晓松先生的这段微博：

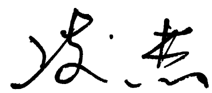
> @高晓松
> 作为自幼学棋，崇拜国手的业余棋手，看了 Master50 : 0 横扫中日韩顶尖高手的对局，难过极了。为所有的大国手伤心，路已经走完了。多少代大师上下求索，求道求术，全被破解。未来一个八岁少年只要一部手机就可以战胜九段，荣誉信仰灰飞烟灭。等有一天，机器做出了所有的音乐和诗歌，我们的路也会走完。
> 1 月 4 日 16:21 来自 iPhone 7 Plus

其实之所以会有这样恐惧，大部分是因为人们在讨论人工智能的时候容易将机器“人格化”，很多科幻作品就是这么干的，这看起来很合理，但问题是机器无论如何都不是人。对于机器来说，围棋说穿了不过是一种基于统计学概率的决策模型，属于数学领域的问题，它本来就是机器的强项。用围棋对于人类的难度来推导机器智能的进步，其实是很没有逻辑的事情。而且事实上，今天所流行的这些人工智能方法都是在 20 世纪 70 年代前后提出的理论，今天的辉煌主要是由于硬件的进步为实现提供了基础，但在智能上并没有多大的实质突破。要知道，人们对于鉴定人工智能的主要标准早有定论，那就是图灵测试。

图灵测试关注的是人机对话能力，换句话说，什么时候机器能通过对话骗到你的一百块钱，也比它下棋下赢世界冠军更智能点。而想要增强人机对话能力，自然语言处理就是

首当其冲的一个领域了。正如我们所说，机器的专长是数学领域，所以自然语言处理问题的目的就是要把我们人类的文本、音频转换成可被分析的数学模型，这对于机器来说是比围棋困难得多的事情。这也是人类和机器的根本区别，对于这两种智能来说，困难的定义是截然不同的。

说实话，刚开始译这本书的时候，我对它的翻译难度有些估计不足，很多专业词汇国内还似乎还没有标准译法。有些甚至根本找不到对应的中文翻译。虽然对于每个小节我都期望查阅大量的资料，尽量保证翻译的质量，但实在有点太累人了，太费时了，妥协、遗憾在所难免。在这里向读者们致歉，还希望你们多多包涵。同时也感谢人民邮电出版社的陈冀康编辑对于我拖稿行为的容忍，其实我还想再拖上半年的。



2017 年 1 月 10 日

于新安江畔

前言

这是一本介绍 NLTK 库，以及如何将该库与其他 Python 库搭配运用的书。NLTK 是当前自然语言处理（NLP）社区中最为流行、使用最为广泛的库之一。NLTK 的设计充分体现了简单的魅力。也就是说，对于大多数复杂的 NLP 任务，它都可以用寥寥几行代码来实现。

本书的前半部分从介绍 Python 和 NLP 开始。在这部分内容中，你将会学到一些通用的预处理技术，例如标识化处理（tokenization）、词干提取（stemming）、停用词（stop word）去除；一些专属于 NPL 领域的预处理技术等，如词性标注（part-of-speech tagging）；以及大多数文本相关的 NLP 任务都会涉及的命名实体识别（Named-entity recognition，简称 NER）等技术。然后，我们会逐步将焦点转到更为复杂的 NLP 任务上，例如语法解析（parsing）以及其他 NLP 应用。

本书的后半部分则将更侧重于介绍如何构建一些 NLP 应用，如对于文本分类，可以用 NLTK 搭配 scikit-learn 库来进行。我们还会讨论一些其他的 Python 库，你应该了解一下这些与文本挖掘或自然语言处理任务相关的库。另外，也会带你看看如何从网页和社交媒体中采集数据，以及如何用 NLTK 进行大规模的文本处理。

本书所涵盖的内容

第 1 章　自然语言处理简介。这一章将会涉及一些 NLP 中的基本概念，并对 NLTK 和 Python 做一些介绍。这一章的重点是让你快速了解 NLTK，并介绍如何安装所需要的库，以便开始构建一个非常基本的单词云实例。

第 2 章　文本的歧义及其清理。这一章将会讨论在任何文本挖掘和 NLP 任务中所需的所有预处理步骤。这一章将会具体讨论断词处理、词干处理、停用词去除等技术。并且，还会为你详细介绍一些别的文本清理技术，以及如何用 NLTK 来简化它们的实现。

第 3 章　词性标注。这一章将重点对词性标注进行概述。在这一章中，我们将会为你介绍如何将 NLTK 运用到一些标注器中，并讨论 NLTK 中有哪些不同的 NLP 标注器可用。

第 4 章　文本结构解析。这一章将会带你继续深入 NLP，讨论不同的语法解析方法，并介绍如何用 NLTK 来实现这些方法。在此过程中，我们会讨论语法解析在 NLP 语境中的，以及一些常见的信息提取技术（如实体提取）中的重要性。

第 5 章　NLP 应用。这一章将会谈及各种不同的 NLP 应用，我们将会带领你利用一些当前已掌握的知识来构建出一个简单的 NLP 应用实例。

第 6 章　文本分类。这一章将会介绍一些机器学习领域中常见的分类方法。讨论重点将主要集中在文本语料库，以及如何用 NLTK 和 scikit 来构建管道，从而实现一个文本分类器。当然，也会讨论与文本聚类和主题模型相关的内容。

第 7 章　Web 爬虫。这一章将讨论 NLP、数据科学和数据收集中其他方面的处理任务，以及如何从最大的文本数据源之一——Web 中获取相关的数据。在这里，我们将学习如何用 Python 库、Scrapy 来建立一只运作良好的 Web 爬虫（crawler）。

第 8 章　NLTK 与其他 Python 库的搭配运用。这一章将会谈及一些骨干的 Python 库，如 NumPy 和 SciPy。另外，我们也会简单地介绍一下用于数据处理的 panda 和用于可视化处理的 matplotlib。

第 9 章　Python 中的社交媒体挖掘。这一章将致力于数据采集相关的内容。在这里，我们将会讨论社交媒体，以及与社交媒体相关的其他问题。当然，我们也会讨论具体应该如何收集、分析并可视化社交媒体中的数据。

第 10 章　大规模文本挖掘。这一章将讨论如何扩展 NLTK，并配合一些别的 Python 库，使其适应大数据时代规模化执行的需要。我们将会给出一个简短的演示，以说明 NLTK 和 scikit 是如何与 Hadoop 搭配使用的。

前期准备

在阅读这本书之前，我们建议你应该准备好下列软件：

章	所需软件（版本）	自由软件/专有软件	软件下载链接	硬件技术指标	所需操作系统

1～5	Python/Anaconda、NLTK	自由软件	`https://www.python.org/`
			`http://continuum.io/downloads`
			`http://www.nltk.org/`
				通用 UNIX 打印系统	不限

6	scikit-learn、gensim	自由软件	`http://scikit-learn.org/stable/` `https://radimrehurek.com/gensim/`	通用 UNIX 打印系统	不限
7	Scrapy	自由软件	`http://scrapy.org/`	通用 UNIX 打印系统	不限

8	NumPy、SciPy、pandas 以及 matplotlib	自由软件	`http://www.numpy.org/`
			`http://www.scipy.org/`
			`http://pandas.pydata.org/`
			`http://matplotlib.org/`
				通用 UNIX 打印系统	不限

9	Twitter Python API 与 Facebook Python API	自由软件	`https://dev.twitter.com/overview/api/twitter-libraries` `https://developers.facebook.com`	通用 UNIX 打印系统	不限

本书的适用读者

只要你是 NLP 和机器学习领域的爱好者，无论之前有没有文本处理方面的经验，这本书都是为你准备的。当然，这本书也非常适合那些想要快速学习一下 NLTK 的资深 Python 程序员。

编写体例

在本书中，我们会用不同的文本样式来突显不同类型信息之间的区别。下面，我们就通过几个例子来介绍一下这些样式，以及它们所代表的含义。

对于正文当中所涉及的代码、数据库表名、文件夹名、文件名、文件扩展名、路径名、

伪 URL、用户输入以及 Twitter 句柄，我们将采取如下形式：“我们需要创建一个名为 NewsSpider.py 文件，并将其路径设置为/tutorial/spiders。”

接下来是 Python 代码块：

```
>>>import nltk
>>>import numpy
```

还有一般性的代码块：

```
add FILE vectorizer.pkl;
add FILE classifier.pkl;
```

另外，在第 7 章中，我们还将会用到 Scrapy shell 中的 IPython 记法，其样式如下：

```
In [1] : sel.xpath('//title/text()')
Out[1]: [<Selector xpath='//title/text()' data=u' Google News'>]
```

最后是所有命令行输入或输出信息的样式：

```
# cp /usr/src/asterisk-addons/configs/cdr_mysql.conf.sample
     /etc/asterisk/cdr_mysql.conf
```

> **提示：**
> 这种形式表达的是一些需要读者警惕的或需要重点关注的内容。

> **小技巧：**
> 这种形式所提供的是一些提示或小技巧。

读者反馈

我们始终欢迎任何来自读者的反馈信息。它能让我们了解你对于这本书的看法——无论是喜欢还是不喜欢。这些反馈对于我们的选题开发来说都是至关重要的。

对于一般的反馈，你只需简单地给 feedback@packtpub.com 发一份电子邮件，并在邮件的标题中注明这本书的书名即可。

如果你对某一话题有专长，并且有兴趣写（或奉献）一本这方面的书，请参考我们的作者指南：www.packtpub.com/authors。

客户支持

你一直都是 Packt 图书的主人，我们将会尽一切努力来帮助你获取最好的图书资讯。

实例代码的下载

你可以在 http://www.packtpub.com 自己的账户页面中找到所有已购买的 Packt 图书，并下载相关的实例代码。如果你是在别处购买了我们的图书，也可以通过访问 http://www.packtpub.com/support 注册有关文件，我们会通过电子邮件将其直接发给你。

勘误

尽管我们已经尽了最大的努力来确保书中内容的正确性，但错误始终是存在的。如果你在我们的书中发现了错误——无论是关于文字的还是代码的——只要你能告诉我们，我们都将不胜感激。因为这样可以大大减少其他读者在阅读方面所遇到的困难。因此，当你发现错误时，只需要访问 http://www.packtpub.com/submit-errata，选择相应的书名，然后单击“errata submission form”链接并输入相关错误的详细信息即可。一旦你提供的信息获得了确认，相关的内容就被更新到我们的网站或对应图书勘误章节下面现有的勘误表中。

如果想要查看先前已提交的勘误信息，你只需访问 https://www.packtpub.com/books/content/support，并在其搜索域中输入相关图书的名称，所需信息就会出现在下面的勘误部分中。

版权

在互联网上，版权对于所有媒介而言一直是一个很大的问题。在 Packet，我们向来对于版权许可非常重视。如果你在网络上发现任何形式的我们出版过的作品，都请马上将网址或网站名称告知我们，以便于我们采取补救措施。

请将你怀疑有侵权行为的文档链接发送到：copyright@packetpub.com。

你付出的帮助是对作者权利的保护，我们也由此才能继续为你带来有价值的内容。

如有疑问

如果你对本书有任何疑问，也可以通过 questions@packtpub.com 跟我们联系，我们会竭尽所能地帮你解决问题。

目录

第 1 章
自然语言处理简介

现在，让我们先从介绍自然语言处理（NLP）开始吧。众所周知，语言是人们日常生活的核心部分，任何与语言问题相关的工作都会显得非常有意思。希望这本书能带你领略到 NLP 的风采，并引起学习 NLP 的兴趣。首先，我们需要来了解一下该领域中的一些令人惊叹的概念，并在工作中实际尝试一些具有挑战性的 NLP 应用。

在英语环境中，语言处理研究这一领域通常被简称为 NLP。对语言有深入研究的人通常被叫作语言学家，而“计算机语言学家”这个专用名词则指的是将计算机科学应用于语言处理领域的人。因此从本质上来说，一个计算机语言学家应该既有足够的语言理解能力，同时还可以用其计算机技能来模拟出语言的不同方面。虽然计算机语言学家主要研究的是语言处理理论，但 NLP 无疑是对计算机语言学的具体应用。

NLP 多数情况下指的是计算机上各种大同小异的语言处理应用，以及用 NLP 技术所构建的实际应用程序。在实践中，NLP 与教孩子学语言的过程非常类似。其大多数任务（如对单词、语句的理解，形成语法和结构都正确的语句等）对于人类而言都是非常自然的能力。但对于 NLP 来说，其中有一些任务就必须要转向标识化处理、语块分解、词性标注、语法解析、机器翻译及语音识别等这些领域的一部分，且这些任务有一大部分还仍是当前计算机领域中非常棘手的挑战。在本书中，我们将更侧重于讨论 NLP 的实用方面，因此我们会假设读者在 NLP 上已经有了一些背景知识。所以，读者最好在最低限度上对编程语言有一点了解，并对 NLP 和语言学有一定的兴趣。

在阅读完本章之后，我们希望读者能掌握以下内容。

- 对 NLP 及其相关概念有个基本的了解。
- 完成 Python 和 NLTK 及其他库的安装。
- 编写一些非常基本的 Python 和 NLTK 代码片段。

如果你从来没有接触过 NLP 这个概念词，我们在下面给你推荐了两本书，请花一些时间阅读一下其中的任何一本——只需要看看它们的前几章即可。另外，你也应该快速浏览一下维基百科上与 NLP 相关的页面。

- 《Speech and Language Processing》，由 Daniel Jurafsky 与 James H. Martin 合著。
- 《Statistical Natural Language Processing》，由 Christopher D. Manning 与 Hinrich Schü　tze 合著。

1.1　为什么要学习 NLP

关于这个问题，我们可以先来看看 Gartner 公司新一轮的趋势报告，你可以很清晰地看到，NLP 技术赫然高居榜首。目前，NLP 已被认为是业界最为稀缺的技能之一。自大数据的概念问世之后，我们所面对的主要挑战是——业界需要越来越多不仅能处理结构化数据，同时也能处理半结构化或非结构化数据的人才。对于我们所生产出来的那些博客、微博、Facebook 订阅、聊天信息、E-mail 以及网络评论等，各公司都在致力于收集所有不同种类的数据，以便建立更好的客户针对性，形成有意义的见解。而要想处理所有的这些非结构化数据源，我们就需要掌握一些 NLP 技能的人员。

身处信息时代，我们甚至不能想象生活中没有 Google 会是什么样子。我们会因一些最基本的事情而用到 Siri；我们会需要用垃圾过滤器来过滤垃圾邮件；我们会需要在自己的 Word 文档中用到拼写检查器等。在现实世界中所要用到的 NLP 应用数不胜数，如图 1-1 所示。

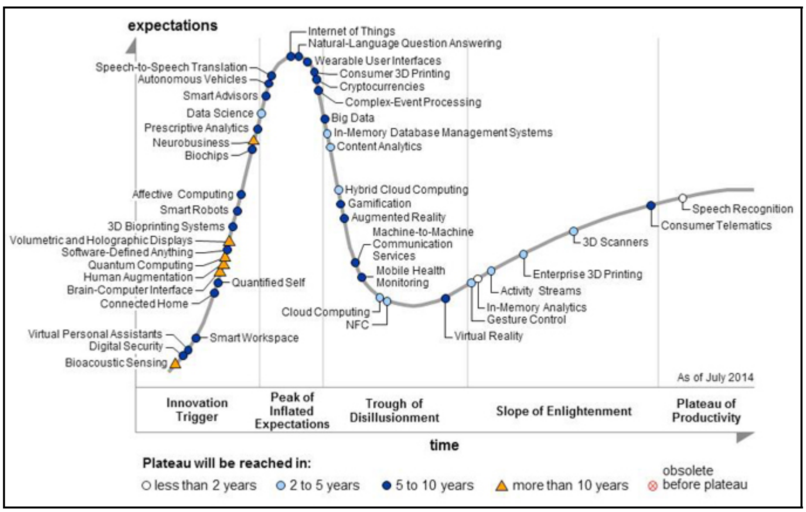


图 1-1

在这里，我们可以再列举一些令人惊叹的 NLP 应用实例。虽然你很可能已经用过它们，但未必知道这些应用是基于 NLP 技术的。

- 拼写校正（MS Word/其他编辑器）。
- 搜索引擎（Google、Bing、Yahoo!、WolframAlpha）。
- 语音引擎（Siri、Google Voice）。
- 垃圾邮件分类（所有电子邮件服务）。
- 新闻订阅（Google、Yahoo!等）。
- 机器翻译（Google 翻译与其他类似服务）。
- IBM Watson[①]。

构建上述这些应用都需要非常具体的技能，需要优秀的语言理解能力和能有效处理这些语言的工具。因此，这些不仅是各 NLP 最具优势领域的未来趋势，同时也是我们用 NLP 这种最独特技能所能创建的应用种类。

在实现上面提到的某些应用以及其他基本的 NLP 预处理时，我们有许多开源工具可用。这些工具有些是由相关组织在建立自己的 NLP 应用时开发的，而有些则纯粹属于开源项目。下面我们就来看一份 NLP 工具的小清单。

- GATE。
- Mallet。
- Open NLP。
- UIMA。
- Stanford toolkit。
- Genism。
- Natural Language Tool Kit (NLTK)。

上述大多数工具都是用 Java 编写的，在功能上也都很相似。尽管这里有一些工具功能很强大，且提供了各种 NLP 实用工具，但如果我们考虑到易用性和其对相关概念的解释度的话，NLTK 的得分就非常高了。NLTK 库是一个非常易学的工具包，这得益于 Python 本身非常平缓的学习曲线（毕竟 NLTK 是用它编写的），人们学习起来会非常快。NLTK 库中收

① 译者注：IBM 最新研制的人工智能系统 Watson，它的运算更快，记忆力也更好，能读懂一些人类语言中的暗喻和双关。

纳了 NLP 领域中的绝大部分任务，它们都被实现得非常优雅，且易于使用。正是出于上述的这些原因，NLTK 如今已成为了 NLP 社区最流行的库之一。

在这里，我们会假设读者已经对 Python 语言有了一定程度的了解。如果你还不了解的话，我们希望你先去学习一下 Python。如今在互联网上可以找到大量的 Python 基础教程，并且能让你对该语言进行一个快速概览的图书也不在少数。当然，我们也会针对不同主题与你探讨 Python 的一些特性。但就目前而言，只要你掌握了基本的 Python 知识，如列表、字符串、正则表达式以及基本的 I/O 操作，就可以继续读下去了。

> **提示：**
> 你可以从下列任意一网站中获取 Python 安装包。
> - `https://www.python.org/downloads/`。
> - `http://continuum.io/downloads`。
> - `https://store.enthought.com/downloads/`。

这里我会推荐读者选用来自 Anaconda 或 Canopy 的 Python 发行版。因为这些发行版本身就具备了一些捆绑库，如 SciPy、numpy、scikit 等，它们可用于数据分析及其他与 NLP 相关领域的应用。甚至，NLTK 也是该发行版的一部分。

> **提示：**
> 请参照下面网址中的说明来安装 NLTK 与 NLTK 数据：
> `http://www.nltk.org/install.html`。

下面，让我们来测试一下。

请在操作系统中打开终端，并运行：

```
$ python
```

该命令应该会为你打开一个 Python 解释器：

```
Python 2.6.6 (r266:84292, Oct 15 2013, 07:32:41)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

我希望你在这里会得到一个与上面情况类似的输出。当然，你也有可能会看到一个不太一样的输出，因此理想情况下，我们应该准备最新版本的 Python（建议是 2.7 版）、GCC

编译器，以及其他操作系统的细部安排。当然，我们知道 Python 目前最新的版本是 3.0 以上，但对于其他任意的开源系统来说，我们应该保守地选择一个更稳定的版本，而不是贸然跳到最新版本。如果你已经将项目迁移到 Python 3.0+，那就务必参阅下面链接中的说明，以便了解那些被添加的新特性：https://docs.python.org/3/whatsnew/3.4.html。

对于基于 UNIX 的系统，Python 属于默认程序（无须任何设置）。而 Windows 用户则需要通过设置相关路径来使 Python 进入正常工作状态。你可以通过以下方式来确认 NLTK 是否已经被正确安装：

```
>>>import nltk
>>>print "Python and NLTK installed successfully"
Python and NLTK installed successfully
```

好了，我们可以准备出发了！

1.2　先从 Python 开始吧

虽然，我们在这里并不打算对 Python 进行任何太过深入的探讨，但带你快速浏览一下 Python 的基础要点还是很有必要的。当然，为了观众着想，我们最好将这次基础性的快速回顾之旅控制在 5 分钟之内。在此期间，我们将会讨论到数据结构的基本知识，一些常用函数，以及在接下来几节中将会用到的 Python 通用结构。

> **提示：**
> 我强烈推荐你花两个小时看一下题为《Google Python class》的参考资料：https://developers.google.com/edu/python，那对我们来说应该算是个不错的开始。当然，你还可以通过 Python 的官方网站 https://www.python.org/来获取更多的教程及其他相关资源。

1.2.1　列表

列表（list）是 Python 中最常用的数据结构之一。它们基本上相当于其他编程语言中的数组。下面，就让我们先从 Python 列表所提供的最重要的那些功能开始吧。

我们可以在 Python 控制台中进行如下尝试：

```
>>> lst=[1,2,3,4]
```

```
>>> # mostly like arrays in typical languages
>>>print lst
[1, 2, 3, 4]
```

当然，Python 列表也可以用更为灵活的索引来进行访问。下面再来看一个例子：

```
>>>print 'First element' +lst[0]
```

在这里，你会得到如下所示的错误信息：

```
TypeError: cannot concatenate 'str' and 'int' objects
```

这是因为 Python 是一种解释型编程语言，它会在对其表达式进行计算的同时检查其中的变量类型。我们在声明这些变量时无需对其进行初始化和类型声明。在这里，我们的列表中所包含的是一些整数对象，它们不能被直接关联到这里的 print 函数上，后者只能接受一个 String 对象。出于这个原因，我们需要将该列表元素转换成字符串。这个过程也称为类型转换。

```
>>>print 'First element :' +str(lst[0])
>>>print 'last element :' +str(lst[-1])
>>>print 'first three elements :' +str(lst[0:2])
>>>print 'last three elements :'+str(lst[-3:])
First element :1
last element :4
first three elements :[1, 2,3]
last three elements :[2, 3, 4]
```

1.2.2 自助功能

如果你想要详细了解 Python 中各种数据类型和函数，最好的方法就是调用其帮助函数，如 help()和 dir(lst)。

其中，我们可以通过 dir（某 Python 对象）命令来列出指定 Python 对象中所有给定的属性。例如，如果我们像下面这样将一个列表对象传递给该函数，它就会列出所有我们可以用列表来做的很酷的事情：

```
>>>dir(lst)
>>>' , '.join(dir(lst))
'__add__ , __class__ , __contains__ , __delattr__ , __delitem__ , __
delslice__ , __doc__ , __eq__ , __format__ , __ge__ , __getattribute__
```

```
, __getitem__ , __getslice__ , __gt__ , __hash__ , __iadd__ , __imul__
, __init__ , __iter__ , __le__ , __len__ , __lt__ , __mul__ , __ne__ ,
__new__ , __reduce__ , __reduce_ex__ , __repr__ , __reversed__ , __rmul__
, __setattr__ , __setitem__ , __setslice__ , __sizeof__ , __str__ , __
subclasshook__ , append , count , extend , index , insert , pop , remove
, reverse , sort'
```

而通过 help（某 Python 对象）命令，我们可以得到给定 Python 对象的详细文档，以及该对象的一些具体用例，如：

```
>>>help(lst.index)
Help on built-in function index:
index(...)
    L.index(value, [start, [stop]]) -> integer -- return first index of value.
This function raises a ValueError if the value is not present.
```

基本上来说，由于 help 和 dir 这两个函数可以运用在任何 Python 数据类型之上，因此它们是一个很好的学习函数和其他对象细节的方法。而且，它还提供了一些基本的使用范例，这在多数情况下都是非常有用的。

Python 的字符串类型与其他语言非常类似，但字符串操作同时也是 Python 最主要的特性之一。即在 Python 中，处理字符串会是一件非常轻松的工作。即使是那些非常简单的操作，例如字符串的切割，你也会看到相较于 Java 和 C 的大费周章，它们在 Python 中是多么得简单明了。

通过之前用过的 help 函数，我们可以得到任何 Python 对象及函数的帮助信息。下面，我们就再来看一些对字符串这种数据类型来说最为常见的操作。

- **split()**：一个能基于某些分隔符来对字符串进行切割的方法。如果你没有为其提供具体参数，它就会默认空格为其分隔符。

```
>>> mystring="Monty Python !  And the holy Grail ! \n"
>>> print mystring.split()
['Monty', 'Python', '!', 'and', 'the', 'holy', 'Grail', '!']
```

- **strip()**：一个可以从字符串中删除其尾随空白符（如'\n'、'\n\r'）的方法。

```
>>> print mystring.strip()
>>>Monty Python !  and the holy Grail !
```

你会注意到'\n'字符被剥离了。另外，你也可以通过 rstrip()和 lstrip()来选择是剥离字符

串左边还是右边的尾部空白符。

- **upper()/lower()**：我们可以用这些方法来改变字符串中字母的大小写。

```
>>> print mystring.upper()
>>>MONTY PYTHON !AND THE HOLY GRAIL !
```

- **replace()**：该方法可用于替换目标字符串中的某个子串。

```
>>> print mystring.replace('!','''''')
>>> Monty Python  and the holy Grail
```

当然，字符串类型的函数可远不止这些。这里只是讨论了其中最常用的一些而已。

> **提示：**
> 你可以通过下面的链接了解更多字符串函数及其用例：https://docs.python.org/2/library/string.html。

1.2.3 正则表达式

对 NLP 爱好者来说，正则表达式是另一个非常重要的技能。正则表达式（regular expression）是一种能对字符串进行有效匹配的模式。我们会大量使用这种模式，以求从大量凌乱的文本数据中提取出有意义的信息。下面，我们就来整体浏览一下你将会用到哪些正则表达式。其实，我这一生至今所用过的正则表达式无非也就是以下这些。

- （句点）：该表达式用于匹配除换行符\n 外的任意单字符。
- \w：该表达式用于匹配某一字符或数字，相当于[a-z A-Z 0-9]。
- \W（大写 W）：该表达式用于匹配任意非单词性字符。
- \s（小写 s）：用于匹配任意单个空白字符，包括换行、返回、制表等，相当于[\n\r\t\f]。
- \S：该表达式用于匹配单个任意非空白字符。
- \t：该表达式用于匹配制表符。
- \n：该表达式用于匹配换行符。
- \r：该表达用于匹配返回符。
- \d：该表达式用于匹配十进制数字，即[0-9]。

- ^：该表达式用于匹配相关字符串的开始位置。
- $：该表达式用于匹配相关字符串的结尾位置。
- \：该表达式用来抵消特殊字符的特殊性。如要匹配$符号，就在它前面加上\。

下面，我们来看一个用于查找东西的例子。在这里，myString 是要进行相关模式查找的目标字符串对象。字符串的子串搜索是 re 模块中最常见的用例之一。我们可以来看看它是如何实现的：

```
>>># We have to import re module to use regular expression
>>>import re
>>>if re.search('Python',mystring):
>>>    print "We found python "
>>>else:
>>>    print "NO "
```

只要我们执行了以上代码，就会立即收到如下信息：

```
We found python
```

我们还可以使用更多正则表达式模式来进行查找。例如，findall()就是一个常被用于对字符串进行全部模式查找的函数。它会按照给定模式对字符串进行查找，并列出其中所有匹配的对象：

```
>>>import re
>>>print re.findall('!',mystring)
['!', '!']
```

如你所见，myString 中存在着两个“!”实例，findall 返回了这两个对象的列表。

1.2.4　字典

字典（dictionary）也是最常用到的一种数据结构。在其他编程语言中有时也被称为**关联数组/存储**。字典是一种键值索引型的数据结构，其索引键可以是任意一种不可变的类型，例如字符串和数字都经常被用来充当索引键。

字典是被多种编程语言广泛用于实现诸多算法的一种非常便利的数据结构。而且，Python 的字典结构还是所有的这些编程语言中最为优雅的哈希表实现之一。哈希表是一种操作起来非常容易的字典结构，其优势在于，你只需通过寥寥几段代码就可以用它建立起一个非常复杂的数据结构，而同样的任务在其他语言中可能就需要花费更多的时间、写更

多的代码。很显然，程序员们应该花更多时间在算法上，而不是数据结构本身。

下面，我打算用字典结构中常见的一个用例来获取某段既定文本中各单词的出现频率分布。你可以看到，只需短短几行代码，我们就取得了各单词在文本中的出现频率。如果你再用任意其他语言来尝试一下相同的任务，就会明白 Python 是何等得奇妙：

```
>>># declare a dictionary
>>>word_freq={}
>>>for tok in string.split():
>>>    if tok in word_freq:
>>>        word_freq [tok]+=1
>>>    else:
>>>        word_freq [tok]=1
>>>print word_freq
{'!': 2, 'and': 1, 'holy': 1, 'Python': 1, 'Grail': 1, 'the': 1, 'Monty':
1}
```

1.2.5 编写函数

和其他编程语言一样，Python 也有自己的函数编写方式。在 Python 中，函数的定义通常会从关键字 def 开始，后面紧跟着相应的函数名和括号()。而所有类似于其他编程语言中的参数和参数类型的声明都会被放在该括号内。其实际代码部分将会从冒号（:）后面开始，代码的初始行通常会是一个文档字符串（注释），接着是代码的主体部分，最后我们会以一个 return 语句来结束整个函数。下面来看个实例，这个函数实例 wordfreq 的开头是关键字 def，它没有参数，最后以一个 return 语句作为结束。①

```
>>>import sys
>>>def wordfreq (mystring):
>>>    '''
>>>    Function to generated the frequency distribution of the given text
>>>    '''
>>>    print mystring
>>>    word_freq={}
>>>    for tok in mystring.split():
>>>          if tok in word_freq:
>>>                 word_freq [tok]+=1
>>>          else:
>>>                 word_freq [tok]=1
>>>    print word_freq
```

① 译者注：原文如此，但正如你所见，代码中并没有 return 语句。在没有返回值的情况下，python 的函数是不必以 return 结束的。

```
>>>def main():
>>>     str="This is my fist python program"
>>>     wordfreq(str)
>>>if __name__ == '__main__':
>>>     main()
```

如你所见，其代码主体与上一节中所写的完全相同，只不过我们这回以函数的形式使这段代码具备了可重用性和可读性。当然，用解释器风格来编写 Python 代码的做法也很常见，但从大型程序的编写实践来说，使用函数/类和某种成熟的编程范式是一个更佳的做法。而且，我们也希望用户能早日编写并运行自己的第一个 Python 程序。对此，你需要按照以下步骤来进行。

1. 用你喜欢的文本编辑器创建一个空的 Python 文件 mywordfreq.py。
2. 将上面的代码写入或复制到该文件中。
3. 在操作系统中打开命令行终端。
4. 在该终端中执行以下命令：

   ```
   $ python mywordfreq,py "This is my fist python program !!" 。
   ```

5. 最后，你应该会得到以下输出：

   ```
   {'This': 1, 'is': 1, 'python': 1, 'fist': 1, 'program': 1, 'my':1}。
   ```

现在，相信你对 Python 所提供的一些常见的数据结构有了一个非常基本的了解。你已经可以编写出一个完整的 Python 程序，并成功地执行了它。在我看来，这些 Python 引导知识已经足以让你面对本书最初这几章的挑战了。

> **提示：**
> 你还可以通过下面网站中的一些 Python 教程了解更多相关的 Python 命令：
> https://wiki.python.org/moin/BeginnersGuide。

1.3 向 NLTK 迈进

尽管在这里，我们并不打算深入探讨自然语言处理理论，但也会尽快让你实际接触一下 NLTK。因此，我打算先介绍一些 NLTK 的基本用例，这是一个很好的机会，你可以先为今后做类似事情做一些准备。下面，我们会从一个 Python 程序员习惯的处理方式切入，

演示如何用 NLTK 将该方式转换成一个更为高效、可靠、简洁的解决方案。

我们先来看一个纯文本分析的例子。这个例子是我们要从 Python 官方主页上摘取部分内容。

```
>>>import urllib2
>>># urllib2 is use to download the html content of the web link
>>>response = urllib2.urlopen('http://python.org/')
>>># You can read the entire content of a file using read() method
>>>html = response.read()
>>>print len(html)
47020
```

目前，我们还没有得到任何关于该 URL 所讨论话题的线索，所以接下来，我们要先做一次探索性数据分析（EDA）。通常对于一段文本域而言，EDA 可能包含了多重含义，但这里只会涉及其中的一个简单用例，即该文档的主体术语类型。主题是什么？它们的出现频率如何？整个分析过程还会或多或少地涉及一些预处理层面的步骤。我们会试着先用纯 Python 的方式来实现它，然后用 NLTK 再将其实现一次。

我们先要清理掉其中的 html 标签。一种可行的做法是只选取其中的标记，包括数字和字符。如果之前有在工作中使用过正则表达式，你应该可以轻松地将这些 html 字符串转换成一个标记列表：

```
>>># Regular expression based split the string
>>>tokens = [tok for tok in html.split()]
>>>print "Total no of tokens :"+ str(len(tokens))
>>># First 100 tokens
>>>print tokens[0:100]
Total no of tokens :2860
['<!doctype', 'html>', '<!--[if', 'lt', 'IE', '7]>', '<html', 'class="no-
js', 'ie6', 'lt-ie7', 'lt-ie8', 'lt-ie9">', '<![endif]-->', '<!--[if',
'IE', '7]>', '<html', 'class="no-js', 'ie7', 'lt-ie8', 'lt-ie9">',
'<![endif]-->', ''type="text/css"', 'media="not', 'print,', 'braille,'
...]
```

如你所见，上面列出了我们在处理文本内容时用不到的 HTML 标签和其他多余字符。当然，这个任务还有个更为简洁的版本：

```
>>>import re
>>># using the split function
>>>#https://docs.python.org/2/library/re.html
>>>tokens = re.split('\W+',html)
>>>print len(tokens)
```

```
>>>print tokens[0:100]
5787
['', 'doctype', 'html', 'if', 'lt', 'IE', '7', 'html', 'class', 'no',
'js', 'ie6', 'lt', 'ie7', 'lt', 'ie8', 'lt', 'ie9', 'endif', 'if',
'IE', '7', 'html', 'class', 'no', 'js', 'ie7', 'lt', 'ie8', 'lt', 'ie9',
'endif', 'if', 'IE', '8', 'msapplication', 'tooltip', 'content', 'The',
'official', 'home', 'of', 'the', 'Python', 'Programming', 'Language',
'meta', 'name', 'apple' ...]
```

这样看上去已经简洁多了吧？但其实它还可以更简洁一点。在这里，我们所做的努力是尽可能地去除干扰，但那些被清理的 HTML 标记还是会如雨后春笋般地冒出来，而且我们可能也想以单词长度为标准，删除某一特定长度的单词——如说移除像 7、8 这样的元素，因为在目前情况下，这些都只是干扰词。现在，我们要做的不是用 NLTK 来重复相同的任务，完成这些预处理步骤。因为所有的清理工作都可以通过调用 clean_html()函数[①]来完成：

```
>>>import nltk
>>># http://www.nltk.org/api/nltk.html#nltk.util.clean_html
>>>clean = nltk.clean_html(html)
>>># clean will have entire string removing all the html noise
>>>tokens = [tok for tok in clean.split()]
>>>print tokens[:100]
['Welcome', 'to', 'Python.org', 'Skip', 'to', 'content', '&#9660;',
'Close', 'Python', 'PSF', 'Docs', 'PyPI', 'Jobs', 'Community', '&#9650;',
'The', 'Python', 'Network', '&equiv;', 'Menu', 'Arts', 'Business' ...]
```

很酷吧？而且，这无疑让我们的代码更简洁易行了。

下面再来看看如何获得这些术语的频率分布。当然，我们还是要从纯 Python 的方式做起，之后再告诉你 NLTK 的方式。

```
>>>import operator
>>>freq_dis={}
>>>for tok in tokens:
>>>    if tok in freq_dis:
>>>        freq_dis[tok]+=1
>>>    else:
>>>        freq_dis[tok]=1
```

① 译者注：最新版的 NLTK 已经取消了这个函数，并鼓励用户使用 BeautifulSoup 的 get_text()函数，因此对于：

```
clean = nltk.clean_html(html)
```

我们应该将其改成（当然，在此之前还必须导入 bs4 库中的 BeautifulSoup 模块）：

```
soup = BeautifulSoup(html,"lxml")
clean = soup.get_text()
```

```
>>># We want to sort this dictionary on values ( freq in this case )
>>>sorted_freq_dist= sorted(freq_dis.items(), key=operator.itemgetter(1),
reverse=True)
>>>print sorted_freq_dist[:25]
[('Python', 55), ('>>>', 23), ('and', 21), ('to', 18), (',', 18), ('the',
14), ('of', 13), ('for', 12), ('a', 11), ('Events', 11), ('News', 11),
('is', 10), ('2014-', 10), ('More', 9), ('#', 9), ('3', 9), ('=', 8),
('in', 8), ('with', 8), ('Community', 7), ('The', 7), ('Docs', 6),
('Software', 6), (':', 6), ('3:', 5), ('that', 5), ('sum', 5)]
```

由于目标是 Python 的官方主页，Python 和（>>>）解释器符号自然就成了最常用的术语，这也符合该网站给人的感觉。

当然，这个任务还有一个更好用、也更有效的方法，即调用 NLTK 中的 FreqDist()函数。在此，我们可以来看看调用后前相同代码的比对：

```
>>>import nltk
>>>Freq_dist_nltk=nltk.FreqDist(tokens)
>>>print Freq_dist_nltk
>>>for k,v in Freq_dist_nltk.items():
>>>    print str(k)+':'+str(v)
<FreqDist: 'Python': 55, '>>>': 23, 'and': 21, ',': 18, 'to': 18, 'the':
14, 'of': 13, 'for': 12, 'Events': 11, 'News': 11, ...>
Python:55
>>>:23
and:21
,:18
to:18
the:14
of:13
for:12
Events:11
News:11
```

> **小技巧：**
> **下载示例代码**
> 你在 http://www.packtpub.com 中登录你的账户，从中可以下载你所购买的、由 Packt 出版的所有书籍的示例代码。如果你在别处购得此书，也可以在 http://www.packtpub.com/support 上注册相关文件，我们会用 E-mail 将其直接发送给你。

现在，让我们来做一些更时髦的事。我们来绘制这样的一张图，如图 1-2 所示。

```
>>>Freq_dist_nltk.plot(50, cumulative=False)
>>># below is the plot for the frequency distributions
```

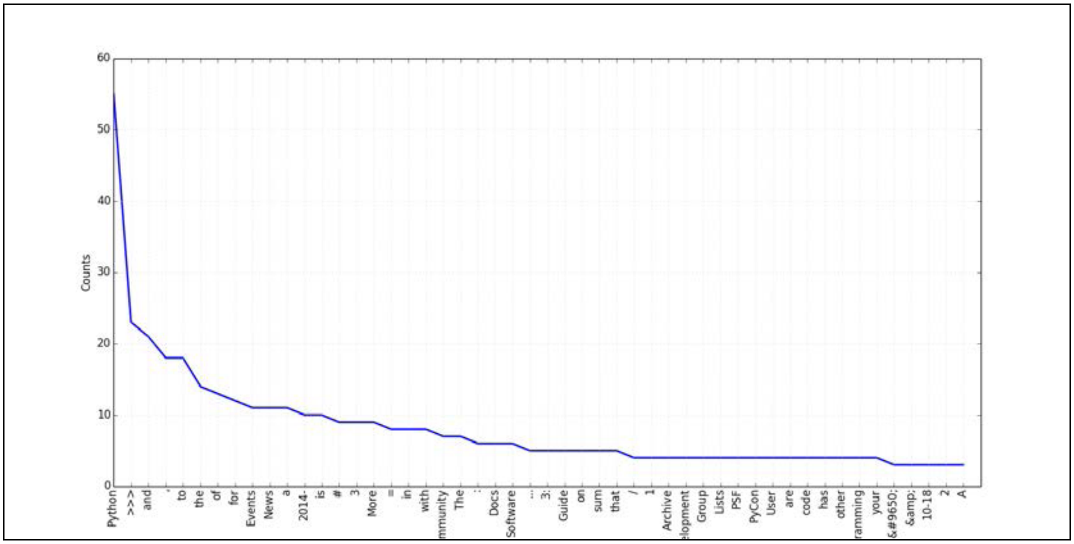


图 1-2

在图 1-2 中，我们可以看到累积频率的即时增长，在某些点上曲线会进入一条长长的尾巴。其中依然存在着一些干扰，有些类似于 the、of、for 以及=这样的词都是属于无用词，这些词有一个专用术语：停用词。如 the、a、an 这样的词也都属于停用词。由于冠词、代词在大多数文档中都是普遍存在的，因而对信息的识别没有帮助。在大多数 NLP 及信息检索任务中，人们通常都会先删除掉这些停用词。下面，让我们再次回到之前运行的那个例子中，绘制结果如图 1-3 所示。

```
>>>stopwords=[word.strip().lower() for word in open("PATH/english.stop.
txt")]
>>>clean_tokens=[tok for tok in tokens if len(tok.lower())>1 and (tok.
lower() not in stopwords)]
>>>Freq_dist_nltk=nltk.FreqDist(clean_tokens)
>>>Freq_dist_nltk.plot(50, cumulative=False)
```

提示：

如果想知道关于词云的更多信息，请访问 http://www.wordle.net/advanced。

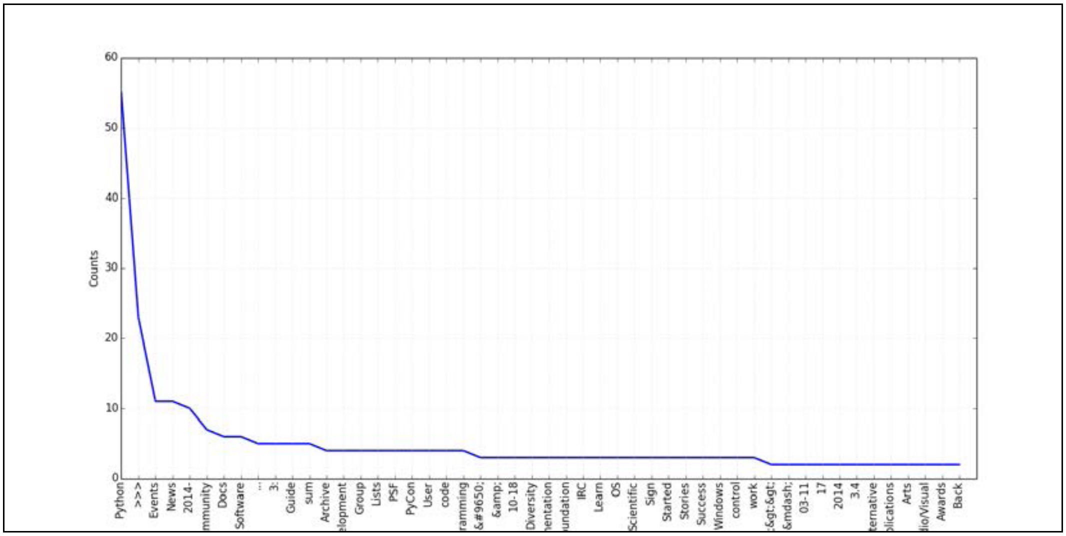


图 1-3

现在，代码看起来简洁多了吧！在完成这么多事后，你可以去 Wordle 网站上将其频率分布以 CSV 形式显示出来，可以得到如图 1-4 所示词云图。



图 1-4

1.4　练习

- 请在不同的 URL 上尝试相同的练习。

- 并试着绘制出相应的单词云。

1.5 小结

总而言之，本章致力于为自然语言处理这一领域提供一份简要概括。虽然，本书假定读者在 NLP 领域，以及使用 Python 编程方面具有一定的背景知识，但我们也提供了一份与 Python 和 NLP 相关的快速入门。我们带你安装了所有在 NLTK 工作中将会用到的程序。另外，我们还通过几行简单的代码给你演示了 NLTK 的使用思路。我们提供的是一个了不起的词云实例，这是在大量非结构化文本中进行可视化处理的一种好方法，同时也是文本分析领域中相当流行的一种运用。我们的目标是要围绕着 NLTK 构建起所需要的一切，并让 Python 在我们的系统上顺利地工作。为此，你也应该要能编写并运行基本的 Python 程序。除此之外，我也希望读者能亲身感受一下 NLTK 库的魅力，自行构建出一个能实际运行的、涉及云词的小型应用程序。只要读者能顺利地产生出云词，我们就认为自己功德圆满了。

在接下来的几章中，我们将更为详细地了解 Python 这门语言，及其与处理自然语言相关的特性。另外，我们还将探讨一些基本的 NLP 预处理步骤，并了解一些与 NLP 相关的基本概念。

第 2 章 文本的歧义及其清理

在上一章中，我们为 Python 以及 NLTK 库的学习开了一个不错的头，带你初步了解了一下如何针对一些文本资料进行一些有意义的 EDA。我们用非常粗糙和简单的方式将预处理部分的所有工作都做了一遍。在本章，我们将具体来讨论**标识化处理**、**词干提取**、**词形还原**（**lemmatization**）以及**停用词移除**等这些预处理步骤。这些话题将会涉及 NLTK 中所有用于处理文本歧义的工具。届时，我们将会讨论现代 NLP 应用中会用到的所有预处理步骤，以及实现其中某些任务的不同方法，并说明我们通常该做什么、不该做什么。总而言之，我们会为你提供关于这些工具的足够信息，以便你可以自行决定在自己的应用程序中使用怎么样的预处理工具。我们希望读者在阅读完本章之后，可以掌握以下内容。

- 所有与数据歧义相关的情况，并能运用 NLTK 处理它们。
- 文本清理的重要性以及我们可以用 NLTK 实现什么样的常见任务。

2.1 何谓文本歧义

事实上，要想给文本/数据歧义这个术语一个定义是相当困难的。本书将它定义成从原生数据中获取一段机器可读的已格式化文本之前所要做的所有预处理工作，以及所有繁复的任务。该过程应该涉及**数据再加工**（**data munging**）、**文本清理**、**特定预处理**、**标识化处理**、**词干提取**或**词形还原**以及**停用词移除**等操作。下面我们就先来看一个基本实例，解析一个 csv 文件：

```
>>>import csv
>>>with open('example.csv','rb')  as f:
>>>    reader = csv.reader(f,delimiter=',',quotechar='"')
```

```
>>>     for line in reader :
>>>         print line[1]    # assuming the second field is the raw sting
```

如你所见，上述代码在试图对 csv 文件进行解析，它将会 csv 文件中所有的列元素构造成一个列表。我们在这操作过程中可以自定义相关的分隔符和引用符（quoting character）。现在的问题是，这些原生字符串会涉及上一章中所学到的那些不同类型的文本歧义。而其中的关键是我们要提供能应付日常 csv 文件的足够细节的信息。

这些最常见文档类型通常都有一个清晰的处理流程，我们可以通过图 2-1 了解一下。

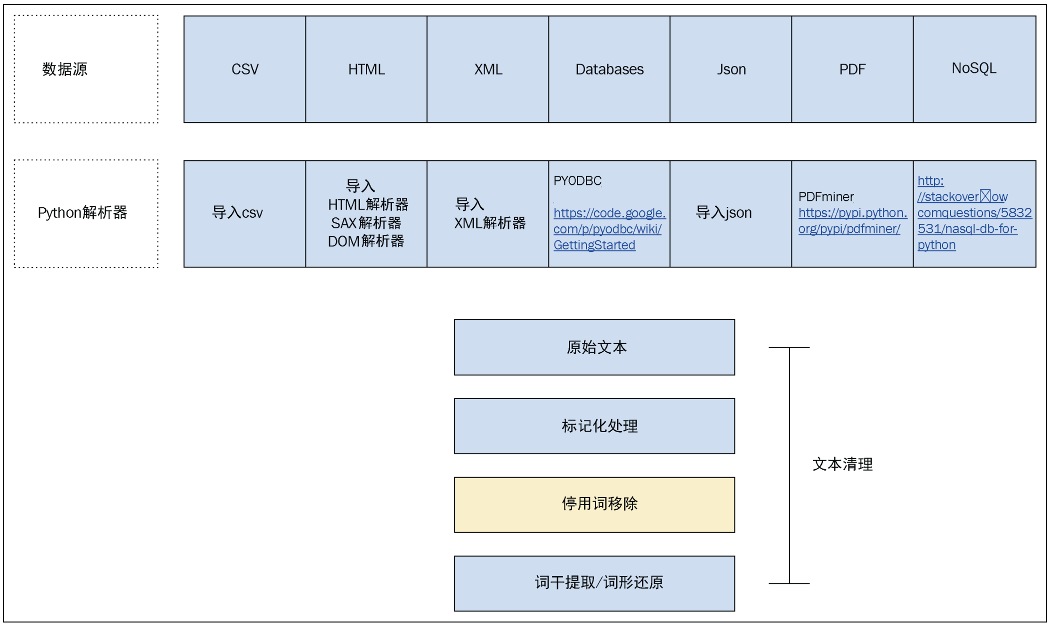


图 2-1

在上图中，堆栈的第一层中列出了一些最常见的数据源。在大多数情况下，我们遇到的数据都属于这些数据格式中的某一个。接下来的这一层是 Python 对于这些数据格式最常见的封装方式。例如在之前的 csv 文件的例子中，Python 的 csv 模块是处理 csv 文件最可靠的方法。通过该模块，我们可以使用各种不同的分离器和引用符等工具。

除此之外，json 也是一种常见的文件格式。

下面来看一个具体的 json 实例：

```
{
  "array": [1,2,3,4],
 "boolean": True,
```

```
  "object": {
    "a": "b"
  },
  "string": "Hello World"
}
```

现在让我们来处理一下该字符串，其解析代码如下：

```
>>>import json
>>>jsonfile = open('example.json')
>>>data = json.load(jsonfile)
>>>print data['string']
"Hello World"
```

如你所见，这里只是用 json 模块加载了一个 json 文件。Python 允许我们挑选相关原生字符串的形式并对其进行处理。关于其他所有数据源的更详细信息以及 Python 中相关的解析工具包，请读者自行参考我们上面列出的那个图表。当然，我们在这里只能指出相关的方向，至于这些工具包的详细信息，还需读者自己上网去搜索。

所以，在我们针对这些不同的文档格式编写自己的解析器之前，请再看一下上图第二行中所列出的 Python 解析器。当我们获得某一段原生字符串时，所有相关的预处理步骤都可以被当作是某一种管道，或者还可以选择性地忽略掉其中的部分内容。下一节，我们将会具体讨论标识化处理、词干提取以及词形还原的相关细节。并且，我们也会讨论一下这些应用的各种变化，以及何时适用于其他场景。

> **提示：**
> 现在，既然我们对文本歧义是什么有了一点想法，就请试着用上述图表中所列出的某个 Python 模块连接任意一种数据库试试。

2.2 文本清理

一旦我们将各种数据源解析成了文本形式，接下来所要面临的挑战就是要使这些原生数据体现出它们的意义。文本清理就泛指针对文本所做的绝大部分清理、与相关数据源的依赖关系、性能的解析和外部噪声等。从这个意义上来说，这些工作和我们在第 1 章——自然语言处理简介中调用 html_clean()对 HTML 文档进行清理的工作是一样的。当然还有

其他情况，如果我们要解析 PDF 文件，可能就需要清理掉一些不必要的干扰字符，移除非 ASCII 字符等。总之在继续下一步骤之前，我们需要做一些清理以获得一个可以被进一步处理的干净文本。而对于像 XML 这样的数据源，我们可能就只需要关注一些特定的树元素即可。对于数据库，我们则有各种可操作的分离器，而且有时我们也只需要关注一些特定的列。总而言之，对于所有致力于净化文本、清理掉文本周围所有可能干扰的工作，我们称之为文本清理。数据再加工（data munging）、文本清理与数据歧义这几个术语之间并没有清晰的界限，它们在类似的语境中可以相互交替使用。在接下来的几节中，我们将会具体讨论一些在任何 NLP 任务中都极为常见的预处理步骤。

2.3 语句分离器

在某些 NLP 应用中，我们常常需要将一大段原生文本分割成一系列的语句，以便从中获取更多有意义的信息。直观地说，就是让语句成为一个可用的交流单元。当然，要想在计算机上实现这个任务可比它看上去要困难得多了。典型的语句分离器既可能是（.）[①]这样简单的字符串分割符，也有可能是某种预置分类器这样复杂的语句边界标识：

```
>>>inputstring = ' This is an example sent. The sentence splitter will split
on sent markers. Ohh really !!'
>>>from nltk.tokenize import sent_tokenize
>>>all_sent = sent_tokenize(inputstring)
>>>print all_sent
[' This is an example sent', 'The sentence splitter will split on
markers.','Ohh really !!']
```

在这里，我们正试着将原生文本字符串分割到一个语句列表中。用的是预处理函数 sent_tokenize()，这是一个内置在 NLTK 库中的语句边界检测算法。当然，如果我们在应用中需要自定义一个语句分离器的话，也可以用以下方式来训练出属于自己的语句分离器：

```
>>>import nltk.tokenize.punkt
>>>tokenizer = nltk.tokenize.punkt.PunktSentenceTokenizer()
```

该预置语句分离器可以支持 17 种语言。我们只需要为其指定相关的配方对象即可。根据我的经验，这里只要提供一个相关种类的文本语料就已经足够了，而且实际上也很少有机会需要我们自己来构建这些内容。

① 译者注：由于原作是基于英文环境来说明的，所以本书的文本处理应该以英文标点为准。

2.4　标识化处理

机器所要理解的最小处理单位是单词（即分词）。所以除了标识化处理之外，我们不宜再对这些文本字符串做更进一步的处理。这里所谓的标识化，实际上就是一个将原生字符串分割成一系列有意义的分词。标识化处理的复杂性因具体的 NLP 应用而异，当然目标语言本身的复杂性也会带来相关的变化。例如在英语中，我们可以通过正则表达式这样简单的方式来选取纯单词内容和数字。但在中文和日文中，这会成为一个非常复杂的任务。

```
>>>s = "Hi Everyone !    hola gr8" # simplest tokenizer
>>>print s.split()
['Hi', 'Everyone', '!', 'hola', 'gr8']
>>>from nltk.tokenize import word_tokenize
>>>word_tokenize(s)
['Hi', 'Everyone', '!', 'hola', 'gr8']
>>>from nltk.tokenize import regexp_tokenize, wordpunct_tokenize, blankline_
tokenize
>>>regexp_tokenize(s, pattern='\w+')
['Hi', 'Everyone', 'hola', 'gr8']
>>>regexp_tokenize(s, pattern='\d+')
['8']
>>>wordpunct_tokenize(s)
['Hi', ',', 'Everyone', '!!', 'hola', 'gr8']
>>>blankline_tokenize(s)
['Hi, Everyone !!  hola gr8']
```

在上述代码中，我们用到了各种标识器（tokenizer）。我们从最简单的——Python 字符串类型的 split()方法开始。这是一个最基本的标识器，使用空白符来执行单词分割。当然，split()方法本身也可以被配置成一些较为复杂的标识化处理过程。因此在上面的例子中，我们其实很难找出 s.split()与 word_tokenize()这两个方法之间的差异。

word_tokenize()方法则是一个通用的、更为强大的、可面向所有类型语料库的标识化处理方法。当然，word_tokenize()是 NLTK 库的内置方法。如果你不能访问它，那就说明在安装 NLTK 数据时出了些差错。请参照第 1 章“自然语言处理简介”中的内容来安装它。

通常情况下，我们有两个最常用的标识器。第一种是 word_tokenize()，这是我们的默认选择，基本上能应付绝大多数的情况。另一选择是 regex_tokenize()，这是一个为用户特定需求设计的、自定义程度更高的标识器。其他的大部分标识器都可以通过继承正则表达式的标识器来实现。我们也可以利用某种不同的模式来构建一个非常具体的标识器。如在

上述代码的第 8 行，我们也可以基于正则表达式的标识器分割出相同的字符串。你可以用\w+这个正则表达式，它会从目标字符串中分隔出所有我们所需要的单词和数字，其他语义符号也可以通过类似的分割器来进行分离，如对于上述代码的第 10 行，我们可以使用\d+这个正则表达式。这样我们就能从目标字符串中提取出纯数字内容。

现在，你能为提取大小写单词、数字和金钱符号构建专用的正则表达式标识器吗？

提示：只需参考之前正则表达式的查询模式来使用 regex_tokenize()即可。

> **小技巧：**
> 你也可以去 http://text-processing.com/demo 这个网站找一些演示项目来参考一下。

2.5 词干提取

所谓词干提取（stemming），顾名思义就是一个修剪枝叶的过程。这是很有效的方法，通过运用一些基本规则，我们可以在修剪枝叶的过程中得到所有的分词。词干提取是一种较为粗糙的规则处理过程，我们希望用它来取得相关分词的各种变化。例如 eat 这个单词就会有像 eating、eaten、eats 等变化。在某些应用中，我们是没有必要区分 eat 和 eaten 之间的区别的，所以通常会用词干提取的方式将这种语法上的变化归结为相同的词根。由此可以看出，我们之所以会用词干提取方法，就是因为它的简单，而对于更复杂的语言案例或更复杂的 NLP 任务，我们就必须要改用词形还原（lemmatization）的方法了。词形还原是一种更为健全、也更有条理的方法，以便用于应对相关词根的各种语法上的变化。

下面，我们就来看一段词干提取的具体过程：

```
>>>from nltk.stem import PorterStemmer # import Porter stemmer
>>>from nltk.stem.lancaster import LancasterStemmer
>>>from nltk.stem.Snowball import SnowballStemmer
>>>pst = PorterStemmer()   # create obj of the PorterStemmer
>>>lst = LancasterStemmer() # create obj of LancasterStemmer
>>>lst.stem("eating")
eat
>>>pst.stem("shopping")
shop
```

一个拥有基本规则的词干提取器，在像移除-s/es、-ing 或-ed 这类事情上都可以达到 70%以上的精确度，而 **Porter 词干提取器**使用了更多的规则，自然在执行上会得到很不错的精确度。

我们创建了不同的词干提取器对象，并在相关字符串上调用了 stem()方法。结果如你所见，当用一个简单实例来查看时，它们之间并没有太大的差别，但当多种词干提取算法介入时，就会看到它们在精准度和性能上的差异了。关于这方面的更多细节，你可以去看看 http://www.nltk. org/api/nltk.stem.html 页面上的相关信息。通常情况下，我们使用的是 Porter 词干提取器，如果是在英语环境中工作，这个提取器已经够用了。当然，还有 **Snowball 提取器**这一整个提取器家族，可分别用于处理荷兰语、英语、法语、德语、意大利语、葡萄牙语、罗马尼亚语和俄语等语言。特别地，我也曾经遇到过可用来处理印地文的轻量级词干提取器：http://research.variancia.com/hindi_stemmer。

> **小技巧：**
> 我们会建议那些希望对词干提取进行更深入研究的人去看看关于所有词干提取器的相关研究 http://en.wikipedia. org/wiki/Stemming①。
> 但是，对大多数用户而言，Porter 和 Snowball 这两种词干提取器就足以应付大量的相关用例了。在现代的 NLP 应用中，人们有时候会将词干提取当作是一种预处理步骤从而将其忽略掉，因此这往往取决于我们所面对的具体领域和应用。在这里，我们想告诉你一个事实，即如果你希望用到某些 NLP 标注器，如词性标注（POS）、NER 或某种依赖性解析器中的某些部分，那么就应该避免进行词干提取操作，因为词干提取会对相关分词进行修改，这有可能会导致不同的结果。
> 当讨论到一般标注器时，我们还会进一步对此展开讨论。

2.6　词形还原

词形还原（lemmatization）是一种更条理化的方法，它涵盖了词根所有的文法和变化形式。词形还原操作会利用上下文语境和词性来确定相关单词的变化形式，并运用不同的标准化规则，根据词性来获取相关的词根（也叫 lemma）。

```
>>>from nltk.stem import WordNetLemmatizer
>>>wlem = WordNetLemmatizer()
```

① 译者注：相应的中文页面为：https://zh.wikipedia.org/wiki/词干提取。

```
>>>wlem.lemmatize("ate")
eat
```

在这里，WordNetLemmatizer 使用了 wordnet，它会针对某个单词去搜索 wordnet 这个语义字典。另外，它还用到了变形分析，以便直切词根并搜索到特殊的词形（即这个单词的相关变化）。因此在我们的例子中，通过 ate 这个变量是有可能会得到 eat 这个单词的，而这是词干提取操作无法做到的事情。

- 现在你能解释词干提取与词性还原之间的区别了吗？
- 现在你能为自己的母语设计一个 Porter 词干提取器（基于规则）了吗？
- 为什么对于中文这样的语言来说，词干提取器是很难实现的？

2.7 停用词移除

停用词移除（Stop word removal）是在不同的 NLP 应用中最常会用到的预处理步骤之一。该步骤的思路就是想要简单地移除语料库中的在所有文档中都会出现的单词。通常情况下，冠词和代词都会被列为停用词。这些单词在一些 NPL 任务（如说关于信息的检索和分类的任务）中是毫无意义的，这意味着这些单词通常不会产生很大的歧义。恰恰相反的是，在某些 NPL 应用中，停用词被移除之后所产生的影响实际上是非常小的。在大多数时候，给定语言的停用词列表都是一份通过人工制定的、跨语料库的、针对最常见单词的停用词列表。虽然大多数语言的停用词列表都可以在相关网站上被找到，但也有一些停用词列表是基于给定语料库来自动生成的。有一种非常简单的方式就是基于相关单词在文档中出现的频率（即该单词在文档中出现的次数）来构建一个停用词列表，出现在这些语料库中的单词都会被当作停用词。经过这样的充分研究，我们就会得到针对某些特定语料库的最佳停用词列表。NLTK 库中就内置了涵盖 22 种语言的停用词列表。

下面就来具体实现一下停用词移除的整个过程，这是一段用 NLTK 来处理停用词的代码。当然，你也可以像第 1 章“自然语言处理简介”中那样创建一个字典，然后通过查找的方法来解决这个问题。

```
>>>from nltk.corpus import stopwords
>>>stoplist = stopwords.words('english') # config the language name
# NLTK supports 22 languages for removing the stop words
>>>text = "This is just a test"
>>>cleanwordlist = [word for word in text.split() if word not in stoplist]
# apart from just and test others are stopwords
['test']
```

在上述代码片段中，我们所做的是和第 1 章“自然语言处理简介”中一样的停用词移除操作，但这里部署的是一个更为简洁的版本。之前，我们是基于查表法来做的。即使在当前情况下，NLTK 内部所采用的仍然是一个非常类似的方法。这里建议使用 NLTK 的停用词列表，因为这是一个更为标准化的列表，相比其他所有的实现都更为健全。而且，我们可以通过向该库的停用词构造器传递一个语言名称参数，来实现针对其他语言的类似方法。

- 在移除停用词的操作中，背后的数学运算是什么？
- 在停用词被移除之后，我们就可以执行哪些 NLP 操作了？

2.8　罕见词移除

这是一个非常直观的操作，因为该操作针对的单词都有很强的唯一性，如说名称、品牌、产品名称、某些噪音性字符（例如 html 代码的左缩进）等。这些词汇也都需要根据不同的 NLP 任务来进行清除。例如对于文本分类问题来说，对名词的使用执行预测是个很坏的想法，即使这些词汇在预测中有明确的意义。我们会在后面的章节进一步讨论这个问题。总而言之，我们绝对不希望看到所有噪音性质的分词出现。为此，我们通常会为单词设置一个标准长度，那些太短或太长的单词将会被移除：

```
>>># tokens is a list of all tokens in corpus
>>>freq_dist = nltk.FreqDist(token)
>>>rarewords = freq_dist.keys()[-50:]
>>>after_rare_words = [ word for word in token not in rarewords]
```

在这里，我们通过调用 `FreqDist()` 函数获取了相关术语在语料库中的分布情况，并选取了其中最稀有的一些词形成了一个列表，并用它来过滤我们的原始语料库。当然，我们也可以对单一文档执行同样的操作。

2.9　拼写纠错

虽然并不是所有的 NLP 应用都会用到拼写检查器（spellchecker），但的确有些用例是需要执行基本的拼写检查的。我们可以通过纯字典查找的方式来创建一个非常基本的拼写检查器。业界也有专门为此类应用开发的一些增强型的字符串算法，用于一些模糊的字符串匹配。其中最常用的是 edit-distance 算法。NLTK 也为我们提供了多种内置了 edit-distance 算法的度量模块。

```
>>>from nltk.metrics import edit_distance
>>>edit_distance("rain","shine")
3
```

我们将会在后续章节中更具体地介绍该模块。我们还会看到拼写检查器最优雅的实现代码之一，它出自 Peter Norvig 之手，这是一段用纯 Python 实现的、非常易于理解的代码。

> **小技巧：**
> 对于所有从事自然语言处理这一工作的，我都会推荐他们去看看下面这个链接中所介绍的拼写检查内容：http://norvig.com/spell-correct.htm。

2.10 练习

下面是一些开放性答案的问题。

- 请尝试用 pyodbc 库访问任意一个数据库。

  https://code.google.com/p/pyodbc/wiki/GettingStarted

- 你能创建一个基于正则表达式的标识器，令其选取的单词只包含大小写字母、数字和金钱符号吗？

  [\w+]将会选取所有的单词和数字，即[A-Z A-Z0-9]，而[\$]则会匹配金钱符号。

- 词干提取和词性还原这两个操作之间的差异是什么？

  词干提取操作更多时候是一套用于获取词干一般形式的规则方法。而词形还原主要考虑的是当前的上下文语境以及相关单词的 POS，然后将规则应用到特定的语法变化中。通常来说，词干提取的操作实现起来较为简单，并且在处理时间上也要明显短于词形还原。

- 你可以为自己的母语设计一个（基于规则的）Porter 词干提取器吗？

  提示：http://Snowball.tartarus.org/algorithms/english/stemmer.html。

- 在完成停用词移除之后，我们还可以执行其他 NLP 操作吗？

  答案是否定的，这是不可能的。所有典型的 NLP 应用，如词性标注、断句处理等，都需要根据上下文语境来为既定文本生成相关的标签。一旦我们移除了停用词，其

上下文环境也就不存在了。

- 为什么在印地文、中文这样的语言中，词干提取器会变得难以实现？

  因为印度语的词法很丰富，而中文则是标识化的难度很高，它们都在符号的标准化上遇到了一定的挑战，因此词干提取器实现起来要困难得多。我们会在后面的章节中详细讨论这些挑战。

2.11　小结

在这一章中，我们讨论了所有与文本内容相关的数据挖掘与数据再加工话题。我们介绍了一些最常见的数据源，并用相关的 Python 包来对它们进行解析。其中，我们深入地探讨了标识化处理，从非常基本的字符串方法到自定义的基于正则表达式的标识器均有所涉及。

另外，我们还讨论了词干提取和词形还原。在这过程中，我们介绍了各种可用的词干提取器类型及它们各自的优缺点。我们还讨论了停用词移除的过程，这个操作的重要性，何时该执行停用词移除以及何时不需要执行它。我们还简单地讨论了如何清除文本中的罕见词，以及执行文本清理的重要性——这里包含了停用词和罕见词，我们会根据它们的频率分布来重点清除。最后，我们还提到了拼写纠错。我们在文本挖掘和文本清理上可以做的事情是无限的。每一种语料库都是一个新的挑战，并且都存在要除去某种新噪音的需要。我们需要花一点时间来了解一下自己的语料库需要执行什么类型的预处理操作，以及应该忽略掉什么东西。

在下一章中，我们将会看到一些与 NLP 相关的预处理，例如词性标注、断句处理以及 NER 等。我们会在下一章的某些开放性问题的提示和答案中作出解释。

第 3 章 词性标注

上一章对自己所要做的所有预处理步骤进行了讨论，以便在工作中可以应对任何文本语料库。我们现在应该可以放心地对任何种类的文本进行解析和清理了。应该执行所有的文本预处理，如针对任意文本的标识化处理、词干提取以及停用词移除等。可以根据自己的需要执行和定制所有相关的预处理工具。到目前为止，已经重点讨论了针对文本型文档的一般性预处理工作。现在，将焦点转向那些动作更为激烈的 NLP 预处理步骤吧。

本章将具体讨论何谓词性标注，以及词性（POS）在 NLP 应用环境中的意义。也会学习如何用 NLTK 标注有意义的信息，并介绍可用于 NLP 密集型应用程序的各种标注器。最后，还将学习如何用 NLTK 来标注命名实体。详细讨论各种 NLP 标注器，并且还会提供一些代码片段来帮助你理解它们。还将会看到这些标注器的最佳实践，以说明在什么地方应该使用哪种标注器。在读完本章之后，读者应了解以下内容。

- 何谓词性标注，以及其在 NLP 中的重要性。
- 如何使用 NLTK 中形形色色的词性标注。
- 如何用 NLTK 创建自定义的词性标注。

3.1 何谓词性标注

其实，我们可能在小时候就已经听说过**词性（POS）**这个术语了，尤其在形容词和副词的实际使用上。要想掌握其中的窍门还是很花时间的。这两者的区别究竟是什么？或许，可以考虑将所有这方面的知识进行编码以创建一个系统。这件事看起来好像挺容易的，但这几十年来，将这些知识转化为可编码的机器学习模型一直都是一个非常难解的 NLP 问题。

在我个人看来，虽然目前最先进的词性标注算法在预测给定单词的词性上已经有了较高的精确度（约 97%），但词性标注领域中仍有大量的研究在等着我们。

对于像英语这样的语言来说，它们在新闻和其他领域往往都有许多已被标注的语料库。这为我们带来了许多先进的算法。尽管在一般情况下，这其中的一些标注器应该足以应付各种跨不同领域的、文本化的使用环境了。但在某些特定的用例中，POS 的预判可能还是有些不尽如人意。对于这些用例，可能就得要从头开始建立一个标注器了。然而，如果想要深入了解 POS，就得先要对机器学习领域中的一些技术有一个基本的了解。虽然这部分有些是要在第 6 章：文本分类中讨论的内容，但在这里，必须先讨论一下相关基础知识，以便可以创建一个自定义的 POS 标注器以满足需求。

首先，我们要学习一些现成可用的 POS 标注器，及其相配的 token 集。在此，会看到一些以**元组**形式存在的、独立单词的 POS。然后，再将焦点转移到其中一些标注其的内部工作原理上。最后，还将讨论如何从头开始创建一个自定义的标注器。

在讨论 POS 时，总少不了会用到 Penn Treebank[①]这个最常用到的 POS 标记库。

标签	相关说明
NNP	专用名词的单数形式
NNPS	专用名词的复数形式
PDT	前置限定词
POS	所有格结束符
PRP	人称代词
PRP$	所有格代词
RB	副词
RBR	相对副词
RBS	最高级副词
RP	小品词
SYM	符号（数学符号或特殊符号）
TO	To

① 译者注：Penn Treebank 原本是一个 NLP 项目的名称，该项目主要用于对相关语料进行标注，标注内容包括词性标注以及句法分析。其语料来源是 1989 年的华尔街日报，包含了 2499 篇文章。这里指代该项目所标注的结果。

续表

标签	相关说明
UH	叹词
VB	动词的基本形式
VBD	动词的过去式
VBG	动词的动名词用法
VBN	动词的过去分词
WP	Wh-代词
WP$	所有格 wh-代词
WRB	Wh-副词
#	井号符
$	美元符
.	句号
,	逗号
:	分号，分隔符
(	左括号
)	右括号
"	直双引号
'	左单引号
"	左双引号
'	右单引号
"	右双引号

这些看起来就是在小学英语课堂上所讲的东西，对吧？现在，既然我们已经了解了这些标签所代表的含义，下面就可以来运行一个实验了：

```
>>>import nltk
>>>from nltk import word_tokenize
>>>s = "I was watching TV"
```

```
>>>print nltk.pos_tag(word_tokenize(s))
[('I', 'PRP'), ('was', 'VBD'), ('watching', 'VBG'), ('TV', 'NN')]
```

如果只想使用新闻类或与其类似语料库的 POS，那么只需要知道前三行代码即可。在这段代码中，先对一段文本进行了标识化处理，然后对其调用了 NLTK 库中的 pos_tag 方法，得到了一组（词形，词性标签）形式的元组。这就是一个 NLTK 库内置的 POS 标注器。

> **提示：**
> 该方法在内部使用的是由 maxent 分类器（我们会在后面章节中讨论分类器这个话题）所训练出来的模型，以预测特定单词属性所属于的类型标签。
> 如果想了解更多其中的细节，可以参考下面这个链接：
> https://github.com/nltk/nltk/blob/develop/nltk/ tag/__init__.py。

由于 NLTK 库使用了 Python 中强大而有效的数据结构，所以在使用由 NLTK 库处理之后所输出的成果时会具有更多的灵活性。

现在，你一定很想见识一下 POS 在真实应用中的典型用法究竟会是怎样的。在一个典型的预处理操作中，通常要尽可能地找出所有的名词。下面来看一段代码，它会找出给定句子中的所有名词：

```
>>>tagged = nltk.pos_tag(word_tokenize(s))
>>>allnoun = [word for word,pos in tagged if pos in ['NN','NNP'] ]
```

现在，试着来回答一下下面的问题。

- 可以在进行词性标注之前先删除掉停用词吗？
- 要如何获取该句子中所有的动词？

3.1.1 Stanford 标注器

NLTK 库还有一个非常棒的特性，就是它还有许多针对其他的预置标注器的封装器，例如 **Stanford 工具包**。下面的 POSTagger 就一个很常见的例子：

```
>>>from nltk.tag.stanford import POSTagger
>>>import nltk
>>>stan_tagger = POSTagger('models/english-bidirectional-distdim. tagger',
'standford-postagger.jar')
```

```
>>>tokens = nltk.word_tokenize(s)
>>>stan_tagger.tag(tokens)
```

> **小技巧：**
> 如果想要使用上述代码，就必须先从下面的链接中获取 Stanford 标注器：
> http://nlp.stanford.edu/software/stanford-postagger-full-2014-08-27.zip。
> 然后将其中的 jar 文件和模型文件解压到一个文件夹中，然后在 POSTagger()的参数中指定其绝对路径。

现在总结一下，用 NLTK 库实现标注任务的方式主要有两种。

1．使用 NLTK 库或其他库中的预置标注器，并将其运用到测试数据上。这两种标注器应该足以应付纯英语文本环境，以及非特殊领域语料库中所有的词性标注任务了。

2．基于测试数据来创建或训练出适用的标注器。这就等于要处理一个非常特殊的用例或者开发一个自定义的标注器了。

下面，来更深入地了解一下一个典型的 POS 标注器在其内部究竟做了些什么。

3.1.2　深入了解标注器

一个典型的标注器通常要用到大量的训练数据，它主要被用于标注出句子中的各个单词，并为其标上 POS 标签。标注是个纯手动的操作，具体如下：

```
Well/UH what/WP do/VBP you/PRP think/VB about/IN the/DT idea/NN of/IN ,/,
uh/UH ,/, kids/NNS having/VBG to/TO do/VB public/JJ service/NN work/NN for/IN
a/DT year/NN ?/.Do/VBP you/PRP think/VBP it/PRP 's/BES a/DT ,/,
```

上面这些样例取自 Penn Treebank 总机的语料库。人们已经针对一些大型语料库做了大量的手动标注工作。甚至还有一个叫作**语言数据联盟（简称 LDC）**[①]组织，那里的人们花了很多时间来研究不同语言的标注、不同的文本种类以及不同的标注操作，如词性标注、句法分析标注，以及对话标注等（后面章节会讨论这些话题）。

---

① 译者注：LDC，全名 Linguistic Data Consortium，这是一个由大学、图书馆、企业、政府、研究机构共同合办的联合企业，成立于 1992 年，目前由宾夕法尼亚大学负责主要运营。LDC 最初的角色只是保存与分发科研要用到的语言数据，后来有了资金，就开始自行收集并构建一些数据，如今该组织已经拥有了非常多的语言数据资源，成为了最主要的科研语言资源管理分发机构之一。

提示：

读者可以从 https://www.ldc.upenn.edu/处获得上述所有的资源以及更多的相关信息。（尽管 LDC 可以免费提供一小部分数据的一小部分，但如果需要也可以去购买整个标注语料库。NLTK 库中大概内置了 PTB 的大约 10%的数据）

如果还希望训练出属于自己的 POS 标注器，就需要针对特定的领域来执行相关的标注操作。这种标注操作就需要有这些领域的专家来协助。

通常情况下，像词性标注这样的标注问题往往会被视为顺序性的标签化问题或者某种分类问题，后者特指人们为特定 token 所生成的正确标签，并用相关判别模型对其进行预判的那一类问题。

为了不让读者直接陷入过于复杂的应用实例中，我们先从一些简单的标注方法开始。

下面来看一段代码，它将会告诉我们 Brown 语料库中各 POS 标签的分布频率：

```
>>>from nltk.corpus import brown
>>>import nltk
>>>tags = [tag for (word, tag) in brown.tagged_words(categories='news')]
>>>print nltk.FreqDist(tags)
<FreqDist: 'NN': 13162, 'IN': 10616, 'AT': 8893, 'NP': 6866, ',': 5133,
'NNS': 5066, '.': 4452, 'JJ': 4392 >
```

如你所见，NN 是这里出现频率最高的标签，我们可以基于此来创建一个非常幼稚的 POS 标注器，用于给所有的测试文本分配 NN 标签。当然，也可以使用 NLTK 中的一个名为 DefaultTagger 的函数来做这件事。DefaultTagger 函数是顺序性标注器，下面就来讨论一下这个标注器。该标注器会去调用 evaluate()函数，该函数主要用于评估相关单词 POS 的准确度。这是针对 Brown 语料库的标注器所用的基准。在 default_tagger 这个案例中，预测准确率在 13%左右。下面，我们将会把同样的基准进一步推广到所有的标注器上。

```
>>>brown_tagged_sents = brown.tagged_sents(categories='news')
>>>default_tagger = nltk.DefaultTagger('NN')
>>>print default_tagger.evaluate(brown_tagged_sents)
0.130894842572
```

3.1.3 顺序性标注器

毫无疑问，上面那个标识器表现不佳。其实，DefaultTagger 本质上只是基类 SequentialBackoffTagger 的一部分而已，后者是一个顺序性的标注服务。标注器会试着基于其所处的上下文环境来模型化相关的标签。而且如果它不能进行正确的标签预测，就会去咨询 BackoffTagger。通常情况下，DefaultTagger 参数都可以被当作一个 BackoffTagger 实体来使用。

下面继续来看一些更复杂的顺序性标注器。

1. N-gram 标注器

N-gram 标注器是 SequentialTagger 的一个子类，它会在其所在的上下文环境中标注出前 n 个单词，并预测给定 token 的 POS 标签。这些标注器中还包含了一些人们的变体，即 UnigramsTagger、BigramsTagger 和 TrigramTagger：

```
>>>from nltk.tag import UnigramTagger
>>>from nltk.tag import DefaultTagger
>>>from nltk.tag import BigramTagger
>>>from nltk.tag import TrigramTagger
# we are dividing the data into a test and train to evaluate our taggers.
>>>train_data = brown_tagged_sents[:int(len(brown_tagged_sents) * 0.9)]
>>>test_data = brown_tagged_sents[int(len(brown_tagged_sents) * 0.9):]
>>>unigram_tagger = UnigramTagger(train_data,backoff=default_tagger)
>>>print unigram_tagger.evaluate(test_data)
0.826195866853
>>>bigram_tagger = BigramTagger(train_data, backoff=unigram_tagger)
>>>print bigram_tagger.evaluate(test_data)
0.835300351655
>>>trigram_tagger = TrigramTagger(train_data,backoff=bigram_tagger)
>>>print trigram_tagger.evaluate(test_data)
0.83327713281
```

其中，基于一元模型的标注将只考虑相关标签的条件频率，以及针对每个给定 token 所能预测到的、频率最高的标签。而 bigram-tagger 参数将会考虑给定的单词和该单词的前一个单词，其标签将以元组的形式来关联被测试单词所得到的标签。类似的，TrigramTagger 参数将让其查找过程兼顾到给定单词的前两个单词。

很明显，TrigramTagger 参数的覆盖范围比较小，而实例精度则高一些。从另一方面来说，UnigramTagger 的覆盖范围则会大一些。为了让准确率与反馈率之间保持一定的平衡，

上述代码片段结合了这三种标注器。首先，它会进行对给定单词的三元模型查找，以预测标签的顺序。前提是如果没有发现其 Backoff 指向 BigramTagger 参数和 UnigramTagger 参数，以及最后的 NN 标签。

2. 正则表达式标注器

接下来，再来看一个顺序性的标注器类，这是一个以标注器为基础的正则表达式。在该类中，我们的任务不再是去寻找确切的单词了。现在，我们可以在定义一个正则表达式的同时定义出给定表达式所对应的标签。例如，在下面的代码中，会看到一些最常见的正则表达式模式是如何获取不同词性的。其中有一些模式都有其各自所关联的 POS 类别。例如，在英语文章中，任何以 *ness* 结尾的词都会是一个形容词。如果不做这样的标注，就需要去写一堆正则表达式和纯 Python 代码来做这件事，而 NLTK 中的 RegexpTagger 参数提供了基于 POS 的模式的这样一种优雅的方式。这种方式也可以被用于所有与 POS 模式相关的领域。

```
>>>from nltk.tag.sequential import RegexpTagger
>>>regexp_tagger = RegexpTagger(
         [( r'^-?[0-9]+(.[0-9]+)?$', 'CD'),   # cardinal numbers
          ( r'(The|the|A|a|An|an)$', 'AT'),   # articles
          ( r'.*able$', 'JJ'),                # adjectives
          ( r'.*ness$', 'NN'),            # nouns formed from adj
          ( r'.*ly$', 'RB'),              # adverbs
          ( r'.*s$', 'NNS'),              # plural nouns
          ( r'.*ing$', 'VBG'),            # gerunds
          (r'.*ed$', 'VBD'),              # past tense verbs
          (r'.*', 'NN')                   # nouns (default)
          ])
>>>print regexp_tagger.evaluate(test_data)
0.303627342358
```

正如你所看到的，我们在这里只是使用了一些基于 POS 的显模式，就能达到大约 30% 的准确度。如果以混合的方式来使用正则表达式标注器，如通过 BackoffTagger，就有可能会使性能得到改善。在预处理步骤中，还有另一个正则表达式标注器的使用用例，就是用它来替换原生的 Python 函数 string.sub()，该类标注器可以用来标注日期模式、资金模式、位置模式等信息。

- 能否修改 N-gram 标注器那一节中 hybrid 标注器的代码，使之成为一个可用的正则表达式标注器？这样做能否改善性能？
- 能否基于标注日期和货币的正则表达式来写一个标注器？

3.1.4 Brill 标注器

Bril 标注器是一种基于转换操作的标注器，其思路是先对给定标签做一个猜测，然后在下一轮迭代中，基于标注器接下来所学到的规则设置返回到原先的错误上并修复它。这也是一种监督型的标注方式，但与 N-gram 标注不同的是，后者会在训练数据中对 N-gram 模式来进行计数，而在这里我们要查找的是转换规则。

如果该标注器是从 Unigram / Bigram 这两个准确度可接受的标注器开始做起的，而后再到 Brill 标注器，我们要查找的就不再是一个三元组，而是一些基于标签、位置以及单词本身的规则。

例如有这样一条规则：

当目标单词的前置词被标注为 TO 时，其标注 NN 就应该被替换为 VB。

在基于 UnigramTagger 打了一些标签之后，如果我们面对的只是一个简单的规则，那就可以去持续地改进这些标注。这种改进是一个交互的过程。通过几次迭代操作和一些更优的规则，Brill 标注器是可以超越某些 N-gram 标注器的。唯一的建议是要注意标注器在训练数据集上被过度配适（over-fitting）的问题。

提示：
读者可以在下面链接中找到更多在工作中被使用的规则实例：
http://stp.lingfil.uu.se/~bea/publ/megyesiBrillsPoSTagger.pdf。

- 你是否能基于自己的观察写出更多的规则？
- 请试着将 Brill 标注器与 UnigramTagger 搭配使用。

3.1.5 基于机器学习的标注器

到目前为止，所用到的都是一些来自 NLTK 或 Stanford 的、用于预先训练的标注器。虽然上一节的一些实例使用了它们，但这些标注器的内部对我们而言仍然是一个黑盒子，例如 pos_tag 内部使用的是**最大熵分类器**（**MEC**），而 StanfordTagger 所采用的也是最大熵算法的一个修改版本。但它们属于不同的模型。其中有许多基于**隐性马尔可夫模型**（**HMM**）和**条件随机场**（**CRF**）的标注器，它们所使用的都是生成性模型。

当然，本书所要介绍的内容无法覆盖上述所有的话题，但我会尽力推荐一些相关的 NLP 类，以便读者对这些概念能有一个大致上的了解。第 6 章：文本分类将会重点介绍一些分类技术，其中也不乏一些非常高级的 NLP 话题，它们需要更多的关注。

如果非要做一个简短说明，我们也可以将这种归类型的词性标注问题看作是一种分类问题，我们会为其指定某个具体的单词以及其他相关的特性，如它的前置词、上下文语境、形态变化等。将给定单词归类到相应的 POS 类别中，而它的其他部分也会根据相似的特性被模型化，以形成某种生成性模型。读者可以参考下面提示框中列出的链接，以查看其中某些话题的资料。

提示：

NLP CLASS：https://www.coursera.org/course/nlp

HMM：http://mlg.eng.cam.ac.uk/zoubin/papers/ijprai.pdf

MEC:

- https://web.stanford.edu/class/cs124/lec/Maximum_Entropy_Classifiers.pdf
- http://nlp.stanford.edu/software/tagger.shtml

3.2 命名实体识别（NER）

除了 POS 之外，找出文本中的实体项也是最常见的标签化问题之一。通常情况下，NER 主要由实体名、位置和组织构成。当然，有些 NER 系统要标注的实体不止这三项。这也可以被视作是一个顺序性的标签化问题，可以利用上下文语境和其他相关特性来标签化这些命名实体。当然，NLP 领域中还有大量的研究在推进当中，人们正试图研究如何标签化生物实体、零售产品实体等。NLTK 库中 NER 标注的方式也主要有两种。一种方式是使用预先训练好的 NER 模型，我们只要读取测试数据的得分即可。另一种方式是建立一个机器学习基本模型。为此，NLTK 库提供了 ne_chunk()方法和一个封装了 Stanford NER 标注器的命名实体识别系统。

NER 标注器

NLTK 库提供的的命名实体提取方法是 ne_chunk()。我们已经用一小段代码给读者演

示了如何用它来对任意语句进行标注。这种方法需要先进行文本的预处理，即先对语句进行标识化处理，然后再进行语块分解和词性标注的处理顺序，之后才能进行命名实体的标注。虽然 NLTK 库在这里会用到 ne_chunking，且并没有拆分任何东西，但它标注了多个分词，以便能通过它们调用到某个有意义的实体项。

对于命名实体来说，NE 语块分解的方式是基本相同的：

```
>>>import nltk
>>>from nltk import ne_chunk
>>>Sent = "Mark is studying at Stanford University in California"
>>>print(ne_chunk(nltk.pos_tag(word_tokenize(sent)), binary=False))
(S
  (PERSON Mark/NNP)
  is/VBZ
  studying/VBG
  at/IN
  (ORGANIZATION Stanford/NNP University/NNP)
  in/IN
  NY(GPE California/NNP)))
```

如你所见，ne_chunking 方法主要用于识别相关的人员（即姓名），以及他所在的地点（即位置）和组织。如果 binary 参数被设置为 True，该方法就会给出整个句子的树结构以及上面的每一个标签。而当该参数被设置为 False 时，该方法就会提供具体的人员、位置和组织的信息，其情况与之前使用 Stanford NER 标注器的例子一样。

和 POS 标注器的情况类似，NLTK 库中也封装了 Stanford NER。该 NER 标注器具有更高的准确度。下面，通过一段代码来看看这种标注器的具体使用方式。你将会在这个例子中看到，只用三行代码就可以标注出所有的实体：

```
>>>from nltk.tag.stanford import NERTagger
>>>st = NERTagger('<PATH>/stanford-ner/classifiers/all.3class.distsim.
crf.ser.gz',...                  '<PATH>/stanford-ner/stanford-ner.jar')
>>>st.tag('Rami Eid is studying at Stony Brook University in NY'.split())
[('Rami', 'PERSON'), ('Eid', 'PERSON'), ('is', 'O'), ('studying', 'O'),
('at', 'O'), ('Stony', 'ORGANIZATION'), ('Brook', 'ORGANIZATION'),
('University', 'ORGANIZATION'), ('in', 'O'), ('NY', 'LOCATION')]
```

如果仔细观察，就会发现即使对一段非常小的测试语句来说，Stanford 标注器的执行性能也要好于 NLTK ne_chunk 标注器。

如今，虽然这些各式各样的 NER 标注器已经成为了一种非常受欢迎的、用来执行各

种通用实体标注的解决方案，但我们还是有必要训练出属于自己的标注器，以应对像生物学、医学这样特定领域的标注实体任务，因此需要创建出属于自己的 NER 系统。当然，我在这里也会向你推荐一个 NER：Calais。它的标注方式不止是典型的 NER，还有更多实体项。而且这个标注器的性能也非常不错，其网址为 https://code.google.com/p/python-calais/。

3.3　练习

下面来回答一下之前章节中提出的问题。

- 能否在执行词性标注之前移除掉停用词？

  不能。如果移除掉这些停用词，就等于丢失了上下文语境，而有一些 POS 标注器（预先训练模型）是要以单词的上下文语境为特征来标出给定单词的 POS 的。

- 如何获取相关语句中的所有动词？

  可以用 pos_tag 来获取该语句中的所有动词：

```
>>>tagged = nltk.pos_tag(word_tokenize(s))
>>>allverbs = [word for word,pos in tagged if pos in
 ['VB','VBD','VBG'] ]
```

- 能否修改 N-gram 标注器那一节中 hybrid 标注器的代码，使之成为一个可用的正则表达式标注器？这样做能否改善性能？

  是的，可以修改 N-gram 标注器那一节中 hybrid 标注器的代码，使之成为一个可用的正则表达式标注器：

```
>>>print unigram_tagger.evaluate(test_data,backoff= regexp_tagger)
>>>bigram_tagger = BigramTagger(train_data, backoff=unigram_ tagger)
>>>print bigram_tagger.evaluate(test_data)
>>>trigram_tagger=TrigramTagger(train_data,backoff=bigram_tagger)
>>>print trigram_tagger.evaluate(test_data)
0.857122212053
0.866708415627
0.863914446746
```

  而且其性能也得到了改善，因为这里加入了一些基于模式的基本规则，替代了预测出现频率最高的标签。

- 能否基于标注日期和货币的正则表达式来写一个标注器？

  是的，我们可以基于标注日期和货币的正则表达式来写一个标注器。其代码如下：

```
>>>date_regex = RegexpTagger([(r'(\d{2})[/.-](\d{2})[/.-](\d{4})$'
,'DATE'),(r'\$','MONEY')])
>>>test_tokens = "I will be flying on sat 10-02-2014 with around
10M $ ".split()
>>>print date_regex.tag(test_tokens)
```

> **提示：**
> 最后两个问题并没有确切的答案。
> 因为这里有许多规则要取决于读者自己的观察，所以并没有正确/错误的答案。

现在，你能否再来挑战一下第 1 章中词云这样的题目？当然，其范围只局限于名词和动词。

参考资料：

https://github.com/japerk/nltk-trainer

http://en.wikipedia.org/wiki/Part-of-speech_tagging

http://en.wikipedia.org/wiki/Named-entity_recognition

http://www.inf.ed.ac.uk/teaching/courses/icl/nltk/tagging.pdf

http://www.nltk.org/api/nltk.tag.html

3.4 小结

本章的主旨是要向读者介绍 NLP 预处理步骤中最有用的标注操作。一般情况下，我们所讨论的词性问题包含了 POS 在 NLP 语境中所具有的意义。当然，我们还具体讨论了在 NLTK 库中各种不同的、用于预先训练 POS 标注器的方式，它们是如此得简单易用，并且能创建出如此精彩的应用。然后，又对所有可用的词性标注选项进行了探讨，如 N-gram 标注、正则表达式标注等。另外，还开发了一些混合型的标注器，以应对一些特定领域的语料库。

其中，我们简单地介绍了一个典型的预先训练标注器的构建过程，探讨了各种可能用

于解决标注问题的方法。最后，还讨论了 NER 标注器，以及该标注器在 NLTK 库中的工作方式。我希望读者在阅读完本章之后，能理解 POS 和 NER 在通用 NLP 语境中的重要性，并且知道如何运行并使用本章中所列出的 NLTK 代码，这样本章的目的就达到了。但旅程并没有就此结束。现在只是了解了一些粗浅的 NLP 预处理步骤，以及在大多数实践应用中 POS、NER 的主要运用。而在问答系统、文本综述和语音应用这一类更为复杂的 NLP 应用中，需要用到语块分解、语法解析、语义学等这些更深层的 NLP 技术，而这些正是下一章将要讨论的话题。

第 4 章
文本结构解析

本章的内容将会让我们对文本的深层结构有一个更好的理解，并掌握解析文本的具体方法，以及如何在不同的 NLP 应用中使用它这些方法。如你所知，我们目前已经完成了 NLP 中的各种预处理步骤，接下来就该进入到一些更深层次的文本处理了。语言的结构是非常复杂的，需要按照其结构处理的各层次来对它进行描述。本章将讲解所有的文本结构，介绍这些结构之间的区别，并详细介绍其中部分结构的具体用法。另外，还将讨论**上下文无关语法**（**context-free grammar**，**简称 CFG**），以及它在 NLTK 库中的具体实现。还会带你浏览各种不同的文本解析器，并介绍如何使用 NLTK 库中现有的一些解析方法。具体而言，会用 NLTK 库来写一个浅解析器，其中将会再次讨论到语块分解语境中的 NER 问题。也会详细地为你介绍 NLTK 库中现有的一些可用于深层文本结构分析的选项。我们会试着为你提供一些关于信息提取的真实用例，以便介绍本章提及的这些话题所发挥的具体作用。总而言之，我们希望读者在阅读完本章之后能对这些话题有一定程度的理解。

本章将会介绍以下内容。

- 首先，会介绍文本解析是什么，以及与 NLP 相关的文本解析究竟是怎样的。
- 然后，会讨论各种不同的文本解析器，以及如何用 NLTK 库来执行解析。
- 最后，还将讨论文本解析在信息提取操作中的作用。

4.1　浅解析与深解析

通常情况下，在深入解析或者全面解析的过程中，像 CFG、PCFG（即 **probabilistic context-free grammar**，**概率性上下文无关语法**）以及搜索策略这样的语法概念的作用都是

要将一套完整的语法结构应用到某个句子上。其中，浅解析（shallow parsing）是一种面向给定文本的，对其语法信息部分所进行的有限解析任务。而深解析（deep parsing）则是一种更为复杂的应用。一般来说，深解析比较适合于对话系统和文本综述这样的应用，而浅解析则更适合于信息提取和文本挖掘这一类的应用。接下来将会用几节的篇幅来讨论它们各自的优缺利弊，以及在 NLP 应用中的具体用法。

4.2 两种解析方法

业界对于文本解析这个话题，主要存在着两种观点/方法，其具体情况如下表所示。

基于规则的方法	基于概率的方法
该方法基于规则和语法。	在该方法中，我们会通过运用概率模型来学习规则和语法。
在该方法中，我们将会基于 CFG 等语法概念来编撰语法规则手册。	该方法使用的是所观测到的相关语言特征的出现概率。
这是一个自上而下的方法。	这是一个自下而上的方法。
该方法中包含了CFG和基于表达式的解析器。	该方法中包含了 PCFG 和 Stanford 解析器。

4.3 为什么需要进行解析

想要回答这个问题，就得再次回到当年在学校学习语法的时候。现在请告诉我，为什么要学习语法？真的需要学习语法吗？答案当然是肯定的！人长大了自然而然就会去学习自己的母语。学习语言的过程通常是这样的：先学习的是少量的词汇，接着是少量的短语语块，再来就是少量的句子。就在学习各个例句的过程中，我们会学到语言的结构。我们的母亲会怎样一次次地纠正我们说错的句子，当我们试图理解一个句子时就会采用这样一个类似的过程。当然，由于这个过程太日常化了，以至于我们从来没有真正重视过，或者说没有仔细地思考过它。也许等下次我们自己去纠正别人的语法时就明白了。

当要编写解析器时，自然会想着要重复上述过程。如果能提出一组可被当作某种模版的规则，这些规则就能按照某种适当的顺序写出句子。另外，也需要将相关的单词分门

别类。这个过程已经讨论过了，记得吗？词性标注的目的就是让我们知道给定单词所属的类别。

如果理解了上述内容，那么就等于已经掌握了游戏规则，知道应该采取什么有效动作和特定步骤了。我们基本上是在追随一个人脑中非常自然的变化过程，并试图将其模拟出来。其中最简单的语法概念要从 CFG 开始切入，这里所需要的只是一组规则和一个术语分词集。

下面，我们将会用非常有限的词汇量和非常通用的规则来写第一个语法：

```
# toy CFG
>>>from nltk import CFG
>>>toy_grammar =
nltk.CFG.fromstring(
"""
  S -> NP VP              # S indicate the entire sentence
  VP -> V NP              # VP is verb phrase the
  V -> "eats" | "drinks" # V is verb
  NP -> Det N   # NP is noun phrase (chunk that has noun in it)
  Det -> "a" | "an" | "the" # Det is determiner used in the sentences
  N -> "president" |"Obama" |"apple"| "coke"  # N some example nouns
    """)
>>>toy_grammar.productions()
```

目前，这一语法概念所能产生的句子数量是很有限的。下面来思考一种情况：如果现在只知道如何将一个名词和一个动词搭配使用，并且这些动词和名词只能来自于上述代码所列出的单词，那么大概可以搭配出这样的例句。[①]

- President eats apple
- Obama drinks coke

现在来了解一下这个过程中究竟发生了些什么。我们在自己的脑海中创建了一个语法概念，它会基于上述规则和所提供的这些词汇来进行文本解析。如果能够正确地完成这个解析，就能够理解例句的含义。

由此可见，在学校里所学到的英文语法规则是有效的。因为很显然，我们仍在使用这些语法，同时也在不断增强它们，理解所有英文句子的都是这一套相同的规则。但是，今天的规则显然不适用于莎士比亚时期所用的文体。

① 译者注：由于原文针对的是英文语法分析，所以这里的例句就不翻译了。

另一方面，同一套语法也可能会构造出一些毫无意义的句子，例如：

- Apple eats coke
- President drinks Obama

当具体涉及某个**语法解析器**（**syntactic parser**）时，事实上本身就有一定的机率在语法上会形成一些毫无意义的句子。因此如果想要获得其中的语义的话，就需要对句子的语义结构有一个更为深入的理解。我们建议读者应该具体查看一下与自己所感兴趣的语言相关的语法解析器。

4.4 不同的解析器类型

解析器通常要先对一个用于表达一组语法规则的输入字符串进行处理，然后构建出一个或多个可用于构成某种语法概念的规则。简而言之，语法是我们用于衡量一个句子结构是否良好的一份规范说明，而解析器则是一个用于解读语法的程序。该程序会通过搜索各种不同的树结构空间，找出给定句子的最佳树结构。下面来看一些现有的解析器，简要地认识一下这些解析器的运作细节及其实际用途。

4.4.1 递归下降解析器

先来看一个最简单的解析形式，称之为递归下降解析（recursive descent parsing）。这是一个自上而下的处理过程，由于该解析器会从左向右读取输入流中的信息，所以它会试图去验证其语法的正确性。该解析器的基本操作是从输入流中读取字符，然后将它们与终端所输入的语法规则说明进行匹配。递归下降解析器会在其得到一次正确匹配时超前查看一个字符，并领先于其输入流的读取指针。

4.4.2 移位-归约解析器

移位-归约解析器（shift-reduce parser）是一种简单的、自下而上的解析器。和所有常见的、自下而上的解析器一样，移位-归约解析器也会试着去寻找一个单词和短语序列，它们一方面对应着语法生成器的右侧，另一方面则会用生成器左侧的内容对其进行替换，直到归简出完整的句子为止。

4.4.3 图表解析器

我们还会将算法设计中的动态规划技术应用到解析问题上。动态规划技术可以将中间

结果保存下来，然后在适当的时候重新启用它们，以便显著地提高效率。这一技术也可以被应用到文本解析中。这样就可以将解析任务分成几部分来解决，并将各部分的结果先储存起来，然后在必要时再来考虑如何将其有效地组合成一个完整的解决方案。这种方法被称为图表解析（chart parsing）。

提示：
如果想更好地了解解析器的相关信息，读者也可以看看下面链接中的例子：
http://www.nltk.org/howto/parse.html。

4.4.4 正则表达式解析器

正则表达式解析器使用的是一个正则表达式，定义该表达式的语法形式是在完成了词性标注的字符串之上构建而成的。该解析器将使用这些正则表达式来解析给定的句子，并为它们生成相应的解析树。下面就具体来看看正则表达式解析器的操作实例：

```
# Regex parser
>>>chunk_rules=ChunkRule("<.*>+","chunk everything")
>>>import nltk
>>>from nltk.chunk.regexp import *
>>>reg_parser = RegexpParser('''
        NP: {<DT>? <JJ>* <NN>*}    # NP
         P: {<IN>}                 # Preposition
         V: {<V.*>}                # Verb
        PP: {<P> <NP>}             # PP -> P NP
        VP: {<V> <NP|PP>*}         # VP -> V (NP|PP)*
   ''')
>>>test_sent="Mr. Obama played a big role in the Health insurance bill"
>>>test_sent_pos=nltk.pos_tag(nltk.word_tokenize(test_sent))
>>>paresed_out=reg_parser.parse(test_sent_pos)
>>> print paresed_out
Tree('S', [('Mr.', 'NNP'), ('Obama', 'NNP'), Tree('VP', [Tree('V',
[('played', 'VBD')]), Tree('NP', [('a', 'DT'), ('big', 'JJ'), ('role',
'NN')])]), Tree('P', [('in', 'IN')]), ('Health', 'NNP'), Tree('NP',
[('insurance', 'NN'), ('bill', 'NN')])])
```

现在，再来看看上述代码所对应的树结构，它的图形化表示如图 4-1 所示。

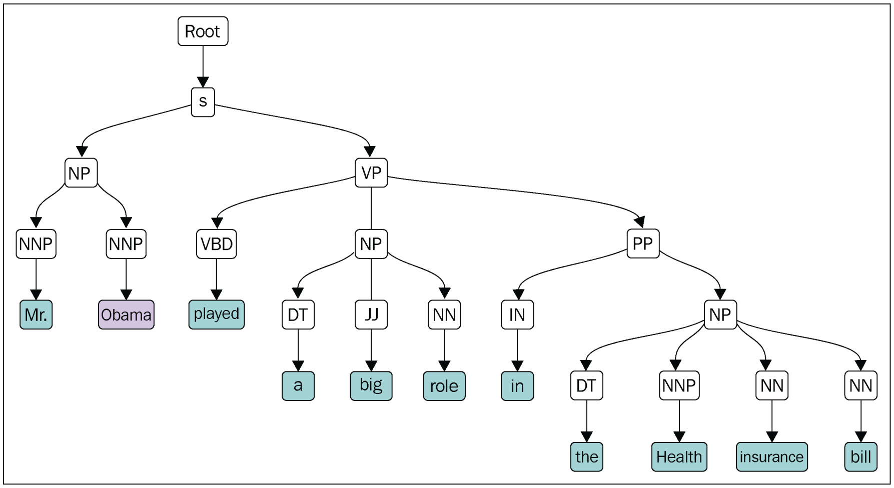


图 4-1

在上述例子中，我们（基于 POS 的正则表达式）定义了模式的种类，希望用模式来生成短语，例如，如果某个短语匹配{<DT>？ <JJ> * <NN>*}这个模式，也就是说它是一个限定词后跟着一个形容词，再加一个名词，那么大多数情况下它会是一个名词性的短语。现在，这更是我们已经定义得到基于规则的解析树的语言规则。得到的是基于规则的解析树。

4.5 依存性文本解析

依存性文本解析（dependency parsing，简称 DP）是一种现代化的文本解析机制。DP 的主要概念是将各个语法单元（单词）用定向链路串联起来。这些链路称为语法上的依存关系（**dependencies**）。在目前的文本解析社区中，有大量这样的工作在进行。尽管短语结构式文本解析（phrase structure parsing）在一些词序自由的语言（例如捷克语和土耳其语）中仍被广泛使用，但依存性文本解析已经被证明是一种更为有效的方法。

在短语结构式文本解析与依存性文本解析之间存在着一个明显的区别，这点可以从它们所产生的解析树上看出来。例如，图 4-2 所示的是“The big dog chased the cat”这句话的解析树。

如果我们仔细看一下这两棵解析树，短语结构树试图捕捉的首先是单词与短语之间的关系，然后再是短语与短语之间的关系。而依存关系树则只关心单词与单词之间的依存关系，例如 big 是完全依赖于 dog 的。

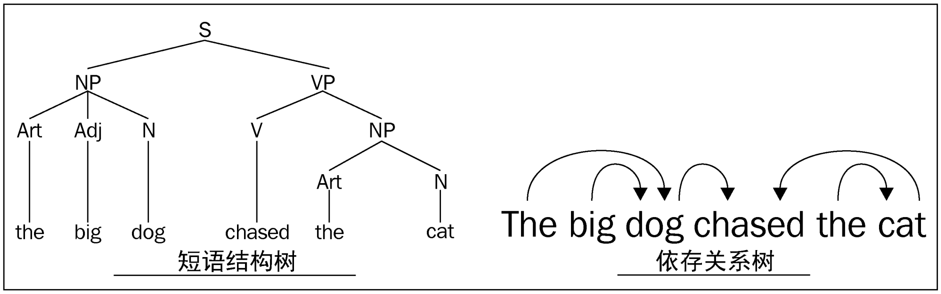


图 4-2

NLTK 库也提供了一些可用于执行依存性文本解析的方法。其中一个方法就是使用**基于概率的投射依存性解析器**（**probabilistic, projective dependency parser**），但该解析器得经由某个有限训练数据集来进行训练。依存性解析器的另一种形态就是 Stanford 解析器。幸运的是，NLTK 库也封装了该解析器。在下面的例子中，就来看看如何使用 NLTK 库中的 Stanford 解析器：

```
# Stanford Parser [Very useful]
>>>from nltk.parse.stanford import StanfordParser
>>>english_parser = StanfordParser('stanford-parser.jar', 'stanfordparser-
3.4-models.jar')
>>> english_parser.raw_parse_sents(("this is the english parser test")
Parse
(ROOT
  (S
    (NP (DT this))
    (VP (VBZ is)
      (NP (DT the) (JJ english) (NN parser) (NN test)))))
Universal dependencies
nsubj(test-6, this-1)
cop(test-6, is-2)
det(test-6, the-3)
amod(test-6, english-4)
compound(test-6, parser-5)
root(ROOT-0, test-6)
Universal dependencies, enhanced
nsubj(test-6, this-1)
cop(test-6, is-2)
det(test-6, the-3)
amod(test-6, english-4)
compound(test-6, parser-5)
root(ROOT-0, test-6)
```

这段输出乍看之下好像很复杂，但其实它一点都不复杂。这是一个三段式的结果列表，其中第一段输出的只是给定句子中的 POS 标签及其解析树。图 4-3 用更为优雅的方式为你呈现了相同的结构。接下来的第二段输出的是给定单词之间的依存关系以及它们各自的位置。最后的第三段输出的是这些依存关系的扩大版：

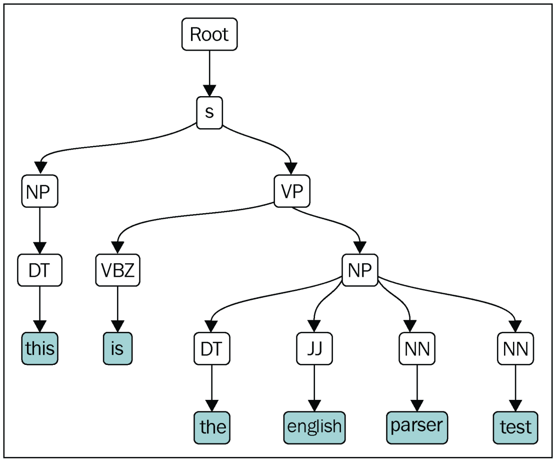


图 4-3

> **小技巧：**
> 如果想对 Stanford 解析器的使用有一个更好的理解，读者可以参考下面链接中的内容：
> http://nlpviz.bpodgursky.com/home
> http://nlp.stanford.edu:8080/parser/index.jsp。

4.6 语块分解

语块分解属于浅解析，它并不会深入到句子的深层结构。在该操作中，我们的目的是尽量将句子分割成一些有意义的构成语块。

将语块定义成文本解析中最小的可处理单元。例如，可以将“the President speaks about the health care reforms”这个句子分解成两个语块。第一个语块是“the President”，该语块由名词主导，所以将其称为**名词短语**（**NP**）。而该句子的其余部分则是一个以动词为主导的语块，因而称为**动词短语**（**VP**）。如果观察得再仔细一些，就会发现“speaks about the health care

reforms”这部分还可以再分出子语块。具体来说就是其中还存在更多的 NP，因而还可以再继续被分解成“speaks about”和“health care reforms”，整个过程如图 4-4 所示。

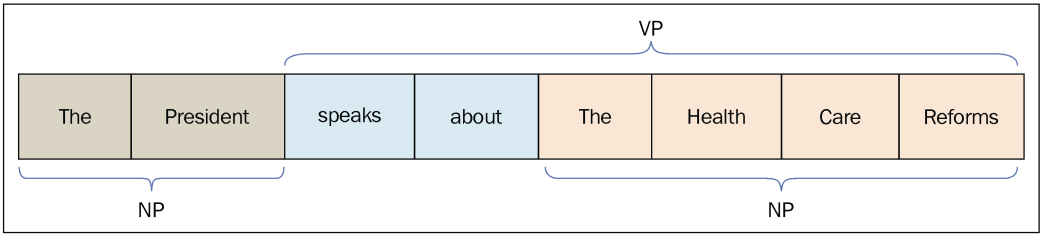


图 4-4

这种将句子划分成各个部分的过程，就是之前所说的语块分解。从形式上来看，语块分解操作也可以被看作是一种处理接口，作用是识别出任意文本中互不重叠的分组。

现在，已经了解了浅解析与深解析之间的区别。当我们能在 CFG 的帮助下进入到句子的语法结构，并理解了这些句子的语法结构时。有些时候就需要用文本解析来理解句子的具体含义了。当然在另一方面，也有一些情况是不需要做如此深度的分析的。比方说，对于大部分非结构化文本，一般只会想要提取其中的关键短语、命名实体或者相关项目的特定模式。在这种情况下，要做的是浅解析而非深解析，因为深解析会去处理所有违反语法规则的句子，也会产生各种不同的语法树，直到解析器在反复回溯的过程中找到最佳的解析树。整个过程非常耗时和繁琐，并且即使完成了所有的这些处理过程，也未必会得到正确的解析树。而浅解析则可以用语块来保证其浅解析结构，这种处理相对而言要较快一些。

下面，我们就写一些代码来做一些基本的语块分解：

```
# Chunking
>>>from nltk.chunk.regexp import *
>>>test_sent="The prime minister announced he had asked the chief government
whip, Philip Ruddock, to call a special party room meeting for 9am on Monday to
consider the spill motion."
>>>test_sent_pos=nltk.pos_tag(nltk.word_tokenize(test_sent))
>>>rule_vp = ChunkRule(r'(<VB.*>)?(<VB.*>)+(<PRP>)?', 'Chunk VPs')
>>>parser_vp = RegexpChunkParser([rule_vp],chunk_label='VP')
>>>print parser_vp.parse(test_sent_pos)
>>>rule_np = ChunkRule(r'(<DT>?<RB>?)?<JJ|CD>*(<JJ|CD><,>)*(<NN.*>)+',
'Chunk NPs')
>>>parser_np = RegexpChunkParser([rule_np],chunk_label="NP")
>>>print parser_np.parse(test_sent_pos)
(S
  The/DT
```

```
    prime/JJ
    minister/NN
    (VP announced/VBD he/PRP)
    (VP had/VBD asked/VBN)
    the/DT
    chief/NN
    government/NN
    whip/NN
  ….
  ….
  ….
  (VP consider/VB)
    the/DT
    spill/NN
    motion/NN
    ./.)

  (S
    (NP The/DT prime/JJ minister/NN)                 # 1st noun phrase
    announced/VBD
    he/PRP
    had/VBD
    asked/VBN
    (NP the/DT chief/NN government/NN whip/NN)         # 2nd noun phrase
    ,/,
    (NP Philip/NNP Ruddock/NNP)
    ,/,
    to/TO
    call/VB
    (NP a/DT special/JJ party/NN room/NN meeting/NN)    # 3rd noun phrase
    for/IN
    9am/CD
    on/IN
    (NP Monday/NNP)                    # 4th noun phrase
    to/TO
    consider/VB
    (NP the/DT spill/NN motion/NN)                     # 5th noun phrase
    ./.)
```

上述代码已经足以应付一些划分动词、名词短语之类的基本语块分解操作了。语块分解的过程中通常会有一条管道，作用是标记 POS 标签，并为相关的语块分解器提供输入字符串。在这里，我们使用的是普通的语块分解器，其中的 NP/VP 规则定义了各种不同的、

可被称之为动词/名词短语的 POS 模式。例如，NP 规则定义的是所有以限定词开头，后接一个由副词、形容词或纯数字的可被语块分解成一个名词短语的组合。当然，这种基于一般表达式的语块分解器得依靠我们手动设计分块字符串来定义分块规则。但如果能编写适用于大部分名词短语模式的普适规则，也可以用基于正则表达式的语块分解器来做这件事。不幸的是，这类普适性的规则是很难找到的。另一种途径是使用机器学习的方法来进行语块分解。之前所接触过的 ne_chunk()方法和 Stanford NER 标注器都使用了同一个预训练模型，可用来标识名词短语。

4.7 信息提取

现在已经学习了标注器和解析器的相关知识。利用这些知识，就可以构建出一个基本的信息提取（IE）引擎了。下面就直接来看一个非常基本的 IE 引擎，看看如何用 NLTK 库来看开发一个典型的 IE 引擎。

任何稍有意义的信息都可以被绘制出来，前提是给定的输入流要跟着 NLP 的步骤来做。我们之前已经对句子标识化、词汇标识化及词性标注有了足够的了解。下面，就来好好讨论一下 NER 和关系提取。

一个典型的信息抽取管道在结构上都是非常类似的，具体如图 4-5 所示。

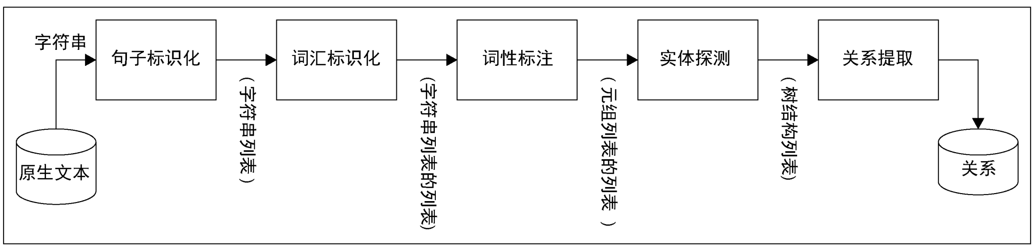


图 4-5



提示：

其他的部分预处理步骤（比如停用词移除和词干提取）往往会被忽略掉，因为它们不会向 IE 引擎中添加任何值。

4.7.1 命名实体识别（NER）

已经在上一章中讨论过了关于 NER 的一般情况。从本质上来说，NER 其实是一种提

取信息的方式，它提取的是一些最常见的实体信息，如实体的名称、所属的组织、以及所在的位置等。然而，某些经改良之后的 NER 也可用于提取一般实体，如产品名称、生物医学项目、作者姓名、品牌名称等。

下面来看一个很普通的例子，在该例子中会给出一个包含既定内容的文本文件，然后再从中提取出一些最有意义的命名实体：

```
# NP chunking (NER)
>>>f=open(# absolute path for the file of text for which we want NER)
>>>text=f.read()
>>>sentences = nltk.sent_tokenize(text)
>>>tokenized_sentences = [nltk.word_tokenize(sentence) for sentence in
sentences]
>>>tagged_sentences = [nltk.pos_tag(sentence) for sentence in tokenized_
sentences]
>>>for sent in tagged_sentences:
>>>print nltk.ne_chunk(sent)
```

在上述代码中，我们只是照着之前图中的相同的管道流程走了一遍。我们执行了所有的预处理步骤，包括句子标识化、词汇标识化、词性标注以及 NLTK 的 NER（预训练模型）等可用来提取所有 NER 的步骤。

4.7.2　关系提取

关系提取也是一种常用的信息提取操作。顾名思义，关系提取就是一个提取不同实体之间不同关系的过程。众所周知，实体之间存在着各种各样的关系，如说我们已经熟悉的继承、同义、近义等关系。这里的关系可以根据信息的需要定义。例如，如果想从非结构化文本数据找出一本书的作者是谁，那么这里的作者身份就可以是作者姓名与书名之间的关系。NLTK 库中也使用了相同的 IE 管道，我们可以沿用 NER 的思路并使用某种基于 NER 标记的关系模式来进行提取。

在下面的代码中，使用了 ieer 的内置语料库，它会对句子进行 NER 标注，这里唯一需要做的是指定所需的关系模式，以及该关系所定义 NER 种类。而且在下面这段代码中，组织与位置之间的关系已经被定义好了，要提取的是这些模式的所有组合。这段代码的应用方式有很多种，例如，在一个非结构化文本的大型语料库中，可以利用相关的位置来识别出感兴趣的组织：

```
>>>import re
>>>IN = re.compile(r'.*\bin\b(?!\b.+ing)')
```

```
>>>for doc in nltk.corpus.ieer.parsed_docs('NYT_19980315'):
>>> for rel in nltk.sem.extract_rels('ORG', 'LOC', doc, corpus='ieer',
pattern = IN):
>>>print(nltk.sem.rtuple(rel))
[ORG: u'WHYY'] u'in' [LOC: u'Philadelphia']
[ORG: u'McGlashan &AMP; Sarrail'] u'firm in' [LOC: u'San Mateo']
[ORG: u'Freedom Forum'] u'in' [LOC: u'Arlington']
[ORG: u'Brookings Institution'] u', the research group in' [LOC:
u'Washington']
[ORG: u'Idealab'] u', a self-described business incubator based in' [LOC:
u'Los Angeles']
..
```

4.8 小结

本章终于越过了基本的预处理步骤，更深入探索了一些 NLP 技术，其中包括了文本解析和信息提取。首先，详细讨论了文本解析技术，介绍了一些可用的解析器，以及如何用 NLTK 库来执行 NLP 中的文本解析。接着，我们带你了解了 CFG 和 PCFG 的概念，以及如何从一个树状语料库学到东西，并构建出一个解析器。最后，还讨论了浅解析与深解析，以及这两者之间的区别。

另外，本章也谈到了一些与信息提取相关的基础知识，包括实体提取和关系提取。我们介绍一个典型的、可充当信息提取引擎的管道。而且正如你所见，用不到 100 行的代码构建了一个非常小巧简单的 IE 引擎。请想象一下，这种系统运行在整个转储程序上，或者将整个网页内容关联到某个组织上时的感觉。这会很酷的，不是吗？

对于本章所学习的这些话题，在未来的章节中用到它们，并利用其中的一些技术构建出一些有用的 NLP 应用。

第 5 章 NLP 应用

这一章要来具体讨论一下 NLP 应用。也就是说，接下来会将用到之前章节所学到的所有概念，看看用这些概念究竟能开发出何种应用程序。因此，这将会是一个完全需要动手实践的章节。在前面的章节中，已经学习了所有 NLP 应用都需要执行的大部分预处理步骤，了解了如何使用标识器、POS 标签、NER 以及如何进行文本解析。本章要提供的是一种思路，让你了解应该如何运用之前所学到的知识开发出一些复杂的 NLP 应用。

如今，现实世界中已经存在着非常多的 NLP 应用程序，如 Google Search、Siri、机器翻译、Google News、Jeopardy[①]和拼写检查等都是一些大家耳熟能详的例子。这其中的一些技术是业界人士多年来的研究成果，他们将这些技术应用到了当前的水平。NLP 太复杂了，正如之前章节中所讲到的那样，像 POS 和 NER 这样的预处理步骤大部分也还都是研究性的问题。但通过使用 NLTK 库，我们已经在恰当的精确度范围内解决了其中的许多问题。本书不会涉及机器翻译和语音识别这样较为复杂的应用。但你现在应该已经具备了足够多的背景知识，也是时候去了解该领域的一些基本应用了。作为一个 NLP 爱好者，我们应该对这些 NLP 应用有一个基本的了解。建议读者可以去互联网上找一些 NLP 应用来看看，并试着去了解它们。

总而言之，本章主要包括以下内容。

- 为读者介绍几个常见的 NLP 应用。
- 利用到目前为止所学习的知识开发一个 NLP 应用（新闻摘要器）。
- 介绍不同 NLP 应用的侧重点，以及它们各自的基本细节。

---

① 译者注：这是一款文字类问答游戏，这些问答题非常考验玩家的英文水平、以及各个领域的知识。玩家要有能力解析题目中的隐晦含义，反讽或者谜题。这也是目前计算机最欠缺的能力。

5.1 构建第一个 NLP 应用

先来看一种非常复杂的 NLP 应用：**信息摘要**（**summarization**）。该应用的概念非常简单：对于所提供的文章、短文、故事，通常会需要针对其内容自动生成摘要。事实上，信息摘要这个应用需要具备一些深层次的 NLP 知识，因为这里需要了解的不单是句子的结构，而是整个文本的结构，除此之外，还得要了解该文本的体裁和主题内容。

鉴于这一切看上去都太过于复杂，所以还是先来尝试一种很直观的方法吧。假设这里所要做的信息摘要是针对句子的重要性和意义进行一次排名。为此，要在理解句子的基础上创建一系列规则，然后用目前为止所学到的处理工具来对新闻文章进行一些可接受的信息摘要处理。

在下面的例子中，我们会将从纽约时报上搜刮来的一篇文章保存在 nyt.txt 这个文本文件中。这里要对这篇新闻稿进行信息摘要。下面就来创建一个个人版的 Google News 吧。

一开始，需要记住一件事：在通常情况下，拥有较多实体和名词的句子的重要性往往会相对比较高。现在的任务是要用某种可被标准化的统一逻辑来计算**重要性评分**（**importance score**）。即如果想获取前 n 个句子的信息情况，就要去选择一个重要性评分的阈值。

现在来看看新闻稿的内容。在这里，你也可以选择将这篇新闻稿以纯新闻内容的形式转储到一个文本文件中。内容具体如下：

```
>>>import sys
>>>f=open('nyt.txt','r')
>>>news_content=f.read()
""" President Obama on Monday will ban the federal provision of some types
of military-style equipment to local police departments and sharply restrict the
availability of others, administration officials said.

The ban is part of Mr. Obama's push to ease tensions between law enforcement
and minority communities in reaction to the crises in Baltimore; Ferguson, Mo.;
and other cities.
- -
blic." It contains dozens of recommendations for agencies throughout the
country."""
```

一旦要对这段新闻内容进行解析，就需整个新闻稿分解成一个句子列表。这样，就回

到了之前讨论过的句子标识器上了，后者会将整个新闻片段分解成若干个句子。在这里，我们会提供一些句型编号，便于识别这些句子并对其进行排名。一旦得到了这些句子，我们会让其在单词标识器中过一遍，最后再来过 NER 标注器和 POS 标注器。

```
>>>import nltk
>>>results=[]
>>>for sent_no,sentence in enumerate(nltk.sent_tokenize(news_content)):
>>>     no_of_tokens=len(nltk.word_tokenize(sentence))
>>>     #print no_of_toekns
>>>     # Let's do POS tagging
>>>     tagged=nltk.pos_tag(nltk.word_tokenize(sentence))
>>>     # Count the no of Nouns in the sentence
>>>     no_of_nouns=len([word for word,pos in tagged if pos in ["NN","NNP"] ])
>>>     #Use NER to tag the named entities.
>>>     ners=nltk.ne_chunk(nltk.pos_tag(nltk.word_tokenize(sentence)),
binary=False)
>>>     no_of_ners= len([chunk for chunk in ners if hasattr(chunk, 'node')])
>>>     score=(no_of_ners+no_of_nouns)/float(no_of_toekns)
>>>
>>>     results.append((sent_no,no_of_tokens,no_of_ners,\
no_of_nouns,score,sentence))
```

在上面的代码中，我们对一个句子列表进行了迭代，并根据公式计算出了这些句子的评分。当然，该公式只是个以被标识实体为分子，以普通标识词为分母的分子式。我们会将所有的这些结果创建成一个元组。

现在，结果就是一个包含了所有评分的元组，例如其中的名词数量、实体数量等。下面来对评分来一个降序排序，代码如下：

```
>>>for sent in sorted(results,key=lambda x: x[4],reverse=True):
>>>     print sent[5]
```

这样一来，就完成了对这些句子的排名。你会为这篇新闻稿能得到这样的结果而感到惊讶。

一旦有了 no_of_nouns 和 no_of_ners 的评分列表，就可以围绕它们去建立一些更复杂的规则。例如，一篇典型的新闻稿通常都会在文章的开头来一个主题说明，并在文章的最后一句也会对整个故事做一个总结。

可以修改之前那段代码，将上述逻辑整合进去吗？

当然，这种信息摘要应用中还包含着另一种理论逻辑：重要的句子中通常包含着重要的词汇，而跨语料库的差异词（discriminatory word）绝大多数都是重要的词汇。因此，只要句子中包含具有很大差异性的词汇，它就是重要的。这样，就得到了一个非常简单的测量方法，就是计算每个词各自的 **TF-IDF**（**term frequency–inverse document frequency**）[1]分值，然后根据词汇的重要性找出一种标准化的平均评分。这个评分就可以用来充当在信息摘要中选取句子的标准。

为了解释清楚概念，这里不会拿整篇文章来举例，这里将只采用文章的前三个句子。下面，就来看看如何用寥寥几行代码实现这个复杂的东西：

> **提示：**
> 这段代码需要你安装一下 scikit 库。如果你已经安装了 anaconda 或 canopy，那么其实就已经安装了这个库，否则请按照下面链接中的指示安装 scikit:
> `http://scikit-learn.org/stable/install.html`。

```
>>>import nltk
>>>from sklearn.feature_extraction.text import TfidfVectorizer
>>>results=[]
>>>news_content="Mr. Obama planned to promote the effort on Monday during a visit to Camden, N.J. The ban is part of Mr. Obama's push to ease tensions between law enforcement and minority \communities in reaction to the crises in Baltimore; Ferguson, Mo. We are, without a doubt, sitting at a defining moment in American policing, Ronald L. Davis, the director of the Office of Community Oriented Policing Services at the Department of Justice, told reporters in a conference call organized by the White House"

>>>sentences=nltk.sent_tokenize(news_content)

>>>vectorizer = TfidfVectorizer(norm='l2', min_df=0, use_idf=True, smooth_idf=False, sublinear_tf=True)

>>>sklearn_binary=vectorizer.fit_transform(sentences)
>>>print countvectorizer.get_feature_names()
>>>print sklearn_binary.toarray()
```

[1] 译者注：TF-IDF 是一种统计方法，主要用来评估某一单词在一个文件集或某个语料库中某一份文件的重要程度。单词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。TF-IDF 加权的各种形式常被搜索引擎应用，作为文件与用户查询之间相关程度的度量或评级。

```
>>>for i in sklearn_binary.toarray():
>>>     results.append(i.sum()/float(len(i.nonzero()[0]))
```

上述代码用到了一些未知的方法，如说 TfidfVectorizer()，这是一个评分方法，它会为给定句子列表中每个句子计算出一个 TF-IDF 评分的向量。现在先别操心这个，我们以后会更详细地来讨论它。本章先暂且将其视为一个黑盒子函数，即对于一个给定的句子/文档列表，它将会给出每个句子所对应的评分，并且还会提供能构建出一个 term-doc 的矩阵，以作为输出。

现在，我们从所有的句子中得到了一个容纳所有单词的字典以及一个评分列表的列表，后者的每一个元素都代表了一个单词所被赋予的 TF-IDF 评分。如果得到的是正确的结果，应该就可以看到一些停用词的评分是接近 0 的，而一些差异词（如 ban 和 Obama）则通常会得到一个很高的评分。一旦在代码中获得了这些数据，就可以通过那些 TF-IDF 值非零的单词来得出 TF-IDF 的平均分值。最后会得到一个与第一个方法类似的评分结果。

你一定会为这个简单的算法能得到这样的结果而感到惊讶。现在，我想我们应该已经做好了所有的准备，可以去编写属于自己的新闻摘要器了。该摘要器会利用上述两种算法对任意两篇给定的新闻稿进行信息摘要处理，并获取不错的摘要结果。虽然这种方法可以实现一个相对还不错的信息摘要程序，但它与信息摘要方面的当前研究水平相比，实际上还差得很远。建议读者多去找一些信息摘要方面的文献来阅读。同时，也希望读者能试着混合使用这两种信息摘要的方法。

5.2　其他 NLP 应用

另外还有一些其他的 NLP 应用，其中包括了文本分类、机器翻译、语音识别、信息检索、信息提取、主题划分和话语分析。这其中有一些问题其实在目前水平下都还是一个非常难以实现的 NLP 任务，相关的领域仍在进行着大量的研究。下一章将会深入地讨论其中的一些话题，但既然是在学习 NLP，就应该先要对这些应用的基本情况有一个了解。

5.2.1　机器翻译

对机器翻译（machine translation）最简单、直接的理解方法就是知道如何将某种语言翻译成另一种语言的。我们会在头脑中对相关句子的结构进行解析，以便试着理解该句子。一旦理解了句子的含义，就会试着将原始语言中的单词替换成目标语言的对应词汇。并且在替换过程中，遵守目标语言的语法规则，以便最终实现正确的翻译。

大致上而言，机器翻译就是图 5-1 所示的那样一个金字塔状的过程。如果以原始语言的文本为出发点，就必须先对目标句子进行标识化处理，后者会被解析成树状结构（简而言之就是它的语法结构），以确保这些句子的正确表达。紧接着就是语义结构，它包含了这些句子所表达的含义，然后，就来到了中间语言这个层面，该中间语言是一种独立于所有语言之外的抽象状态。到目前为止，人们已经开发出了多种翻译方法，这些方法的走向越接近上述金字塔的塔顶，就越需要用到 NLP 技术。因此，根据各种不同的翻译层次，可以使用不同的方法来应对。下面就来看看其中的两种方法。

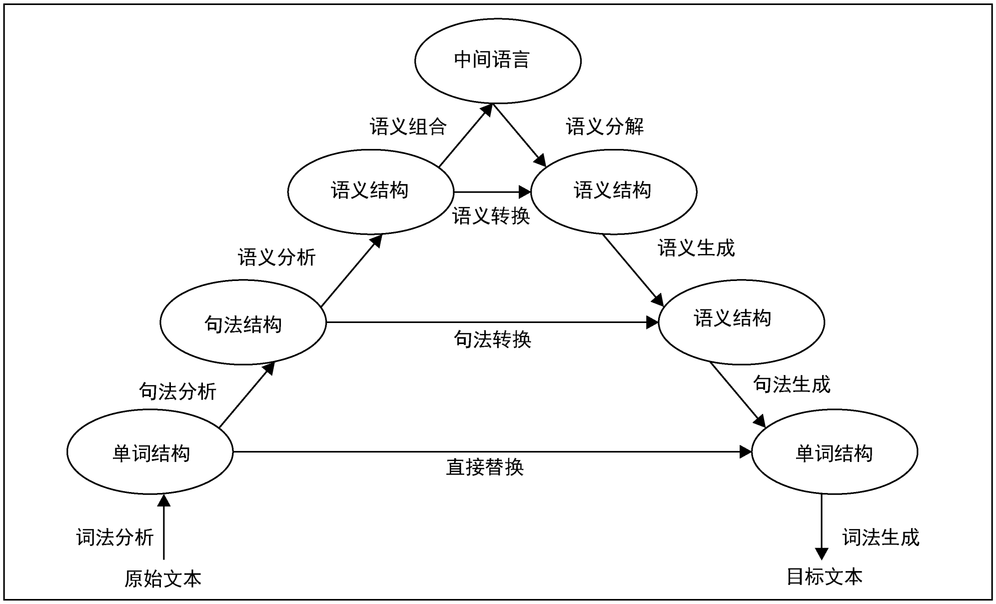


图 5-1

- **直接翻译**：这更像是一种基于字典的机器翻译，当拥有大型语料库以及海量的目标语言词汇时，依赖于相关语言的大型语料库来实现某种类型的翻译应用是可能的。而且因它简单比较流行。
- **语法翻译**：这种翻译方法会试着去构建一个针对原始语言的解析器。到目前为止，人们已经累积了各种各样的关于解析问题的方法。其中有些深层解析器实际上已经拥有了处理一部分语义的能力。一旦解决了解析器，目标词汇的替换问题就迎刃而解了，目标解析器会自行产出最终的目标语言的句子。

5.2.2 统计型机器翻译

统计型机器翻译（Statistical machine translation，简称 SMT）是一种最新型的机器翻译

方法。在这种方法中，人们提出了各种运用统计学的方法，几乎涵盖了机器翻译方面的所有方面。这一类算法背后的思路是依靠所拥有的海量语料库、并行化文本以及可用目标语言产生语言翻译的语言模型。Google Translate 就是一个很好的 SMT 应用实例，它会从不同语言的语料库中学习到相关信息，并围绕这些信息来创建 SMT 应用。

5.2.3 信息检索

信息检索（Information retrieval，简称 IR）也是最受欢迎且被广泛使用的 NLP 应用之一。这类应用最好的实例就是 Google Search，它会根据既定的用户输入，利用信息检索算法检索出与用户查询相关的信息。

简而言之，IR 就是一个根据用户需求来获取最具相关性的信息的过程。在这里系统可以用多种不同的方式来查找信息，当然最终必须要能检索出最具相关性的信息。

典型的 IR 系统的做法是要生成一种索引机制，我们称之为**反向索引（inverted index）**。这类机制与书籍所使用的索引方案非常相似，我们通常会在一本书的最后一页上找到整本书中出现过的单词的一份索引。IR 系统也会创建一份类似的反向索引的 poslist。下面就来看一份典型的 poslist：

```
< Term , DocFreq, [DocId1,DocId2] >
{"the",2 --->[1,2] }
{"US",1 --->[2] }
{"president",2 --->[1,2] }
```

如果有任何单词同时出现在文档 1 和文档 2 中，出处列表中就会有一份指向该单词的文档列表。一旦搞定了数据结构的类型，就可以将其应用到不同的检索模型中。不同的检索模型操作的是不同数据类型。下面就来介绍其中的几个。

1. 布尔检索

在布尔模型中，只需要在 poslist 上执行某种布尔操作即可。例如，如果正在执行类似于“US president”这样的搜索查询，系统就应该去查询“US”和“president”这两个单词 poslist 的交集。

```
{US}{president}=> [2]
```

在这里，第二个文档被证明是具有相关性的文档。

2. 向量空间模型

向量空间模型（vector space model，简称 VSM）的概念来自于几何学。它会以可视化

的方式将文档以向量的形式表示在高维度的词汇空间中。因此，每个文档在该空间中都可以各自用一个向量来表示。虽然向量的表示方式各不相同，但 TF-IDF 无疑是其中最为实用而有效的方式之一。

对于给定的单词和语料库，它们的**单词频率**（**TF**）和**逆文档频率**（**IDF**）的计算公式如下：

$$\mathrm{tf}(t,d)=0.5+\frac{0.5\times \mathrm{f}(t,d)}{\max\{\mathrm{f}(w,d):w\in d\}}$$

在这里，TF 指的只是单词在文档中的出现频率。而 IDF 则指的是文档频率的反比值，也就是该语料库中出现该单词的文档数的累计值：

$$\mathrm{idf}(t,D)=\log\frac{N}{|\{d\in D:t\in d\}|}$$

虽然这些公式的标准变化各种各样，但可以将它们合二为一，创建一个更可靠的评分机制来为文档中的各个单词评分。想要获得 TF-IDF 评分，还需要将这两个评分相乘：

$$\mathrm{tfidf}(t,d,D)=\mathrm{tf}(t,d)\times \mathrm{idf}(t,D)$$

在 TF-IDF 中，我们所评分的是一个单词在当前文档中出现的次数和它在语料库间的散播程度。这让我们认识到一个事实，即跨语料库且出现频率高的单词并不常见。因此可以区别检索这些文档。上一节也使用了 TF-IDF，用来描述信息摘要器。相同评分的文档可用一个向量来表示。一旦将所有文档都表示成了某种向量形式，接下来就可以制定向量空间模型了。

在 VSM 中，用户的搜索查询也被当作是一种文档并表示成一个向量。直观地看，通过计算这两种向量间的点积可以获得目标文档与用户查询之间的余弦相似度。

在图 5-2 中，可以看到这些相同的文档可以用各单词轴线来表示，查询 Obama 与 D1 的相关性要高于 D2。因此该查询的文档相关性评分可表示如下：

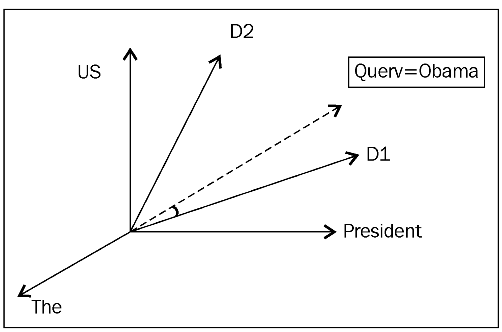


图 5-2

$$\text{sim}(d_j, q) = \frac{\mathrm{d_j} \cdot \mathrm{q}}{\|\mathrm{d_j}\| \|\mathrm{q}\|} = \frac{\sum_{i=1}^{N} w_{i,j}, w_{i,q}}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2} \sqrt{\sum_{i=1}^{N} w_{i,q}^2}}$$

3．概率模型

概率模型会试着去评估相关文档被用户所需要的概率。该模型会假设这个相关概率取决于用户查询和文档表示方式。主要思路是某文档出现在相关性集合中，但不存在于非相关性集合中。下面，用 dj 来表示文档，q 表示用户的查询，R 表示文档的相关性集合，P 表示非相关性集合。那么这里的评分计算应该如下：

$$\text{sim}(d_j, q) = \frac{P(R \mid \vec{d}_j)}{P(\bar{R} \mid \vec{d}_j)}$$



提示：
如果想了解更多关于 IR 的话题，推荐你可以阅读一下下面链接中的内容:
http://nlp.stanford.edu/IR-book/html/htmledition/irbook.html。

5.2.4 语音识别

语音识别是一个非常古老的 NLP 问题。人们自第一次世界大战时代以来就一直在尝试着解决这个问题，目前它还仍然是计算领域最热门的话题之一。这个问题的思路其实很直观。即需要将一段给定的某个人的语音转换成文本。与语音相关的问题都需要生成一个声音序列，我们称之为**音素**（**phonemes**），这是非常难以处理的，因此语音分割本身就是一个大问题。只要语音是可处理的，那么下一步就可以通过一些可用的训练数据来形成某种约束（模型）了，这里就涉及重度的机器学习应用了。如果用图示法将这个应用约束表示成某种方框，就会发现这个方框应该会是整个系统中最复杂的组件之一。其中，它的声学建模涉及的是基于音素的建模，而词汇模型所要解决的模型问题是基于小型的句子分段来进行的，它要将各个分段的含义关联起来。单独的语言建模都是基于单元词法和二元词法来进行的。

一旦构建好了这些模型，就会用它们来处理相关的语句表达。当这些初始化处理过程完成之后，就会对这些语句进行那些声学的、词汇的以及语言建模上的处理，产生相关的

标记并输出，如图 5-3 所示。

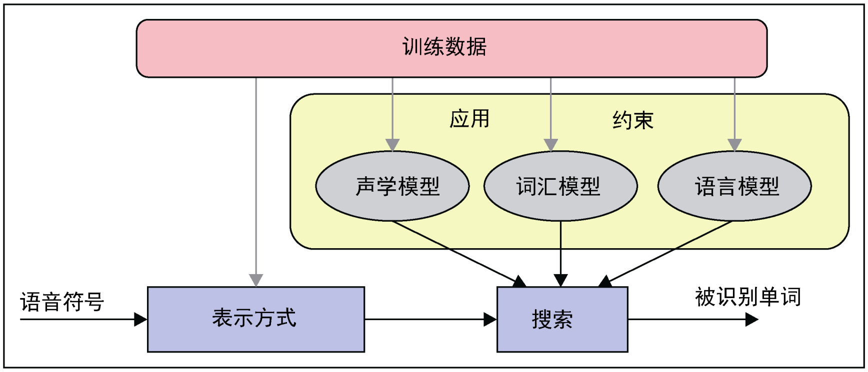


图 5-3

5.2.5 文本分类

文本分类是 NLP 问题中非常有趣且常见的应用。在日常工作中，我们会需要与许多文本分类器进行交互。例如所使用的垃圾邮件过滤器、优先收件箱、新闻聚合器等，所有的这些都是用文本分类技术所构建的应用。

文本分类是一个定义明确且部分已经得到解决的问题，已被用于多个领域。一般情况下，所谓文本分类本质上就是一个利用单词或者词组对文本文档进行分类的过程。虽然这是一个典型的机器学习问题，但文本分类中所用到的许多预处理步骤都来自 NLP 问题。

下面，就来看看文本分类应用的抽象图，如图 5-4 所示。

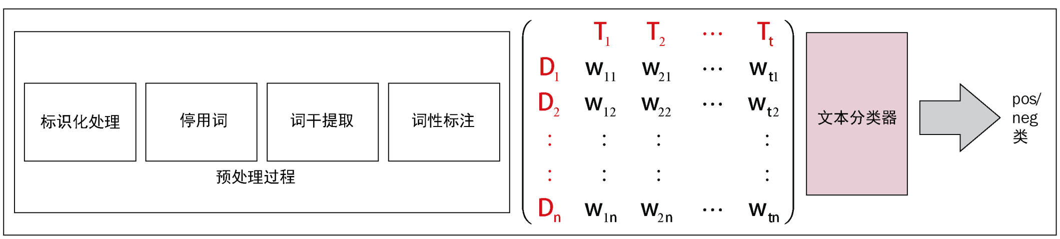


图 5-4

如图所示，我们手里现在有的一堆文档组成了一个类集合。在这里为了让事情简单一点，只讨论了用二进制来进行 1/0 分类的情况。即假设这是在解决一个垃圾邮件检测的问题，用 1 来表示垃圾邮件，而 0 则表示不被视为垃圾邮件的正常文本。

在这个处理过程中，会涉及一些之前章节中所介绍的预处理步骤。其中有些步骤的必要性取决于要解决的是何种类型的文本分类问题。因此在某些少见的情况下，我们会需要根据某特征工程的情况来放弃部分的预处理步骤。该特性工程的最终目标是要生成一个**词汇文档矩阵**（**Term doc matrix**，**简称 TDM**），其中保存的是整个语料库的词汇表，它的列和行都是文档。该矩阵所表现的是一种评分机制，用的是**词袋**（**Bag of word**，**简称 BOW**）表示法。可以通过加权方案将其转变为 TF、TF-IDF、Bernoulli 等其他词频表示法。

除此之外，还可以通过如给定特征的 POS、上下文语境的 POS 等其他方式来进行特征导入。以获取比 NLP 本身更大的特征空间。一旦生成了相关的 TDM，文本分类问题就变成了典型的监督型/非监督性分类问题：即对于给定的一组样本，要预测它们各自的分类。下一章会专门讨论这个话题。这绝对是 NLP/ML 领域中的一个辉煌应用，并且往往都是商业级的应用。

其中有一些还是日常生活中最常见的应用，例如情绪分析、垃圾邮件分类、电子邮件分类、新闻分类、专利分类等。下一章会更具体地讨论文本分类的问题。

5.2.6　信息提取

信息提取（**IE**）是从非结构化文本中提取有意义的信息的过程。同时，IE 也是一种被广泛使用且非常重要的应用。通常情况下，信息提取的引擎会利用海量的非结构化文档来生成某种结构化/半结构化的**知识库**（**knowledge base**，**简称 KB**），然后再围绕着该知识库来部署、构建相关的应用。举个简单的例子，用一组海量的非结构化的文本文档来生成一个非常不错的知识本体（ontology）。Dbpedia 就是这样的一个项目，Wikipedia 中所有的文章都被用作了产生知识本体的构件，这些构件之间都有着一些相互联系或者其他形式的关系。

信息提取主要有两种方式：

- **基于规则的提取**：该方法会用一种模板填充机制。它的思路是要为所预期的结果去寻找一些被预定义好的用例，并试着从该特定的非结构化文本中挖掘出特定的模板。例如，如果要构建足球知识库，需要获取所有球员的信息，其中会涉及他们的简介、统计数据以及部分个人信息等。所有的这些信息都可以被先定义好，然后用基于模式的规则或 POS 标签、NER 和关系提取法来提取。
- **基于机器学习的提取**：这也是一种方法，一种更深度的基于 NLP 的方法，如可以专门针对自己的知识库需求来构建一个解析器。某些知识库将会需要挖掘一些不能

用预训练的 NER 来提取的实体，因此有必要构建一个自定义的 NER。在这种情况下，我们可能会想要去开发一个专门针对待构建知识库的关系提取算法。这会是一种更密集使用 NLP 技术的方法，因为这里要开发的是一个基于 NLP 的解析器或标记器，属于重度的机器学习应用。

5.2.7 问答系统

问答（Question answering，简称 QA）系统是一种基于自身的知识库来解决相关问题的智能系统。这方面最主要的一个例子就是 IBM Watson，因为它参加了电视节目 *Jeopardy*，并且在比赛中赢了人类对手。QA 系统可被分解成若干个构成组件，其中包括用于查询知识库的语音识别，以及用于生成知识库的信息检索和信息提取。

一旦对系统提出了一个问题，对其进行各种不同的分门别类就是它面临的一大问题。另一方面是它要对知识库进行有效地搜索并能检索出最确切的文档。甚至在这之后，还必须通过其他的一些应用（例如信息摘要和解析）以自然的方式得出答案。

5.2.8 对话系统

对话系统一直被当作是一种梦幻般的应用，当该系统收到某种既定源语言的语音时，它会自动执行语音识别，将其转换成文本。然后，该文本会被传递给某种机器翻译系统，该系统可以将该语音的文本翻译成目标语言。接下来，会用某种文本转语音系统将其结果再转换成目标语言的语音。这就是最为理想化的 NLP 应用之一，因为只要有了它，就可以用任何语言与计算机进行通信，同时计算机也将用相同的语言来进行回复。这样一来，这种应用就等于破除了世界上所有现存的语言障碍。

其中，Apple Siri 和 Google Voice 是对话系统中商业应用的典型例子，它们的智能程度都足以了解我们的信息需求，并且都能试着用一组动作和信息来解决这些需求，作出与人类相似的反应。

5.2.9 词义消歧

词义消歧（Word sense disambiguation，简称 WSD）也是一个人们研究多年但仍未得到解决的困难挑战，同时它也是问答系统、信息摘要和搜索等应用所面临的主要难点之一。理解这个概念有一个简单的方法：即就是当遇到不同的上下文语境时，有许多单词的具体含义是不一样的。例如下列例句中都有“cold”这个单词。

- The ice-cream is really cold

- That was cold blooded!

但这两个“cold”的含义并不相同，这种区别的概念对于计算机来说确实是很难理解的。词性标注和 NER 这些 NLP 预处理选项也只能解决其中的部分问题。

5.2.10 主题建模

在大规模非结构化文本内容的处理条件下，主题建模算是一个非常了不起的应用了。它的主要任务是识别出语料库中新出现的主题，然后根据这些主题将其文档分类存放到语料库中。下一章中会对这个问题进行一些简单的讨论。

主题建模应用使用了相同的 NLP 预处理，例如句子分割、标识化处理和词干提取等。该算法的优点是有了一种无人监管的文档分类法;。另外，主题生成的过程并没有明显涉及其他东西。我们鼓励读者更进一步了解主题建模，推荐你去阅读一下与潜在狄氏分配（**latent dirichlet allocation**，简称 LDA）和**潜在语义索引（latent semantics indexing，简称 LSI）**相关的详细信息。

5.2.11 语言检测

对于一段给定的文本来说，对其进行语言检测其实也是一个问题。并且，语言检测的应用对于其他一些 NLP 应用（如搜索、机器翻译、语音等）也非常重要。其主要的想法是要从文本中学习到相关语言的特性。在这个特性工程中，会用到各种与机器学习以及 NLP 相关的技术。

5.2.12 光符识别

光符识别（Optical character recognition，简称 OCR）是一种 NLP 与计算机视觉技术相结合的应用，它会对给定的手写文档/非数字文档进行文本识别，并将其提取为数字格式。这项应用在机器学习领域中也已经被广泛研究了许多年。Google Book 算是其中一个比较大型的 OCR 项目，它们使用 OCR 技术将非数字图书转换到了一个集中式的图书馆中。

5.3 小结

总而言之，我们周围存在许多 NLP 应用，它们充斥着我们的日常交互。NLP 是有一定难度和复杂度的，其中有些问题至今为止也没得到解决，或者没有完美的解决方案。所以每一个探寻 NLP 问题的人都在试着搜索这方面的文献。现在正是成为一名 NLP 研究者的

大好时机。因为在大数据时代，NLP 应用将非常受欢迎。许多研究实验室和组织目前都正在致力于开发像语音识别、搜索和文本分类这一类的 NLP 应用。

相信到现在为止，读者已经学到了许多基础知识。在接下来的几章中，将会深入探讨本章所介绍的一些应用。也就是说，目前已经来到了一个学习节点上，我们已经充分掌握了那些与 NLP 相关的预处理工具，并且对一些最流行的 NLP 应用也已经有了一个基本的了解。下面，希望能利用学到的知识来构建某种版本的 NLP 应用。

下一章将会开始介绍一些重要的 NLP 应用，如文本分类、文本聚类和主题建模。而且，还会稍稍离开一下纯 NLTK 应用程序，去了解一下 NLTK 与其他库的配合使用方法。

第 6 章 文本分类

上一章对 NLP 领域中一些最常见的工具和预处理步骤进行了详细的讨论。本章将会利用之前所学习的大部分知识来构建一种复杂度最高的 NLP 应用。本章将会给出一个解决文本分类问题的通用方法，并带你从零开始用尽可能简短的代码来构建文本分类器。除此之外，还将给出一份适用于文本分类问题的分类算法清单。

虽然本章会对部分最常见的文本算法进行一些讨论，但也都只能是些蜻蜓点水式的介绍，对于那些想要了解具体细节和相关数学背景的读者，本章在后面会列出许多在线资源和相关书籍，以供参考。我们会竭尽所能地帮助读者了解他们所要了解的知识，使读者能够着手使用本章提供的代码片段。尽管，文本分类问题是 NLP 领域中一个很典型的用例，但本章并不打算使用 NLTK 来做这件事，因为 scikit-learn 库中包含了更为广泛的分类算法，使用该库来执行文本挖掘会更为有效。

在阅读完本章之后，希望读者能掌握以下内容。

- 学会所有的文本分类算法并理解它们。
- 学会如何使用点对点的管道来构建文本分类器，并用 scikit-learn 和 NLTK 来实现它。

下面来看一下 scikit-learn 库在机器学习应用上的功能列表，如图 6-1 所示。

事实上，下面这个功能列表可以被当成一个流程图。这样就会有一个明确的方向，如了解哪种方法对应的是哪个问题、分类器之间的迁移要依赖多大规模的标记样本等。对于构建实用程序来说，从这张流程图入手是一个不错的开始，它在大多数情况下都是适用的。本章大多数时候关注的是文本数据，尽管 scikit-learn 库也可以处理其他类型的数据，但在这里只探讨文本（为了降低维度）中的文本分类、文本聚类以及主题检测问题，并构建一些炫酷的 NLP 应用。当然，本章不会对机器学习、分类和聚类的概念进行详细说明，因为对于这些内容，大家可以在 Web 上找到充足的可用资料。我们会在谈到相关语料库时给出

这些概念的更多细节，不过在此之前，还是得先来做个复习。

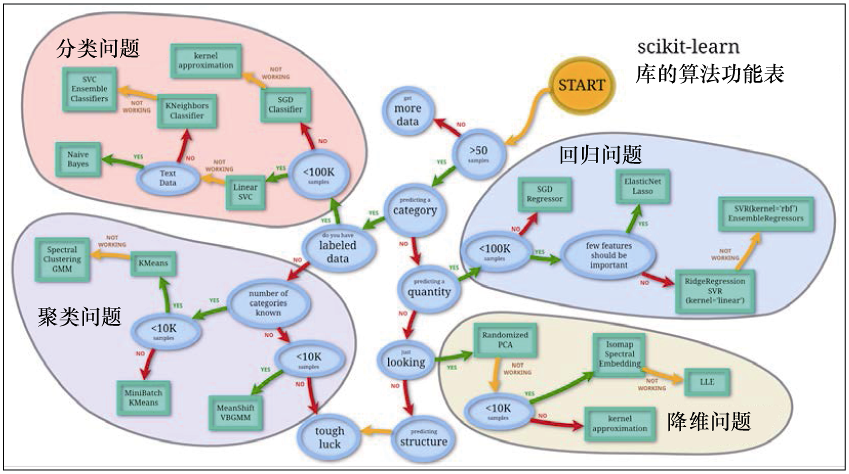


图 6-1

6.1　机器学习

机器学习技术可以被分成两大类型：监督式学习和无监督式学习。

- **监督式学习**：该技术基于若干预先标记的历史样本，它用来预测未知测试样本的算法，主要有以下两类。
  - **分类算法**：该算法主要用于预测测试样本是否属于某些类型中的一个。如果算法中只有两个类，这就是一个二元分类问题；否则就是一个多元分类问题。
  - **回归算法**：该算法主要用于预测某种连续性的变量，例如房价和股票指数等。
- **无监督式学习**：当没有任何标签数据却仍需要预测类标签时，就会用到这种名为无监督式学习的技术。如果需要基于相关项之间的相似性来对它们进行分组，这就是在解决聚类问题。而如果是需要在较低维度上表示高维数据的话，那就更多的是一个降维问题。
- **半监督式学习**：它在分类上应该属于监督式学习任务和技术，但同时也会使用未标记的训练数据。从名称上就可以看出，这更像是一种介于监督式学习和无监督式学习之间的技术，基于少量标记数据和大量未标记数据来构建具有预测能力的机器学习模型。
- **增强式学习**：这是一种利用奖罚机制来实现的机器学习形式，它没有指定的完成任务方式。

如果已经理解了这些不同的机器学习算法，就可以来猜猜看下面这些都属于哪种机器学习问题。

- 下个月的天气预报。
- 从数百万笔交易中检测出欺诈行为。
- Google 的优先收件箱。
- 亚马逊的推荐机制。
- Google 新闻。
- 自动驾驶汽车。

6.2 文本分类

对于文本分类，最简单的定义就是要基于文本内容来对其进行分类。通常情况下，目前所有的机器学习方法和算法都是根据数字/变量特征来编写的。所以这里最重要的问题之一，就是如何在语料库中用数字特征的形式来表示文本。各种技术文献提出了各种不同的转换方式，下面从最简单、使用最广泛的转换方式着手。

为了帮助读者理解文本分类的具体过程，来看看垃圾邮件这个现实问题，毕竟在这个充斥着 WhatsApp 和 SMS 的世界中，我们难免会收到许多垃圾邮件。下面，就来想想如何借助文本分类算法来解决垃圾邮件检测这个现实问题。由于接下来的这个运行实例会贯穿本章的内容，所以需要读者先来手动标记一下这几则真实的 SMS 例文：

```
SMS001 ['spam', 'Had your mobile 11 months or more? U R entitled to
Update to the latest colour mobiles with camera for Free! Call The Mobile
Update Co FREE on 08002986030']
SMS002 ['ham', "I'm gonna be home soon and i don't want to talk about
this stuff anymore tonight, k? I've cried enough today."]
```

> **提示：**
> 读者也可以从下面链接中下载到一份已完成标注的数据集。当然，请确保你创建的是一个和上述例子显示内容相同的 CSV 文件。下面代码中'SMSSpamCollection'所对应的就是这个文件：
> `https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection`。

接下来要做的第一件事是按照之前几章所学到的数据清理、标识化处理以及词干提取等知识来对 SMS 进行清理，使其内容更简洁一些。下面就来写一个基本的、用于文本清理的函数：

```
>>> import nltk
>>> from nltk.corpus import stopwords
>>> from nltk.stem import WordNetLemmatizer
>>> import csv
>>> def preprocessing(text):
>>>     text = text.decode("utf8")
>>>     # tokenize into words
>>>     tokens = [word for sent in nltk.sent_tokenize(text) for word in
nltk.word_tokenize(sent)]

>>>     # remove stopwords
>>>     stop = stopwords.words('english')
>>>     tokens = [token for token in tokens if token not in stop]

>>>     # remove words less than three letters
>>>     tokens = [word for word in tokens if len(word) >= 3]
>>>     # lower capitalization
>>>     tokens = [word.lower() for word in tokens]
>>>     # lemmatize
>>>     lmtzr = WordNetLemmatizer()
>>>     tokens = [lmtzr.lemmatize(word) for word in tokens]
>>>     preprocessed_text= ' '.join(tokens)
>>>     return preprocessed_text
```

第 3 章已经讨论过了与标记化处理、词形还原以及停用词相关的知识。在上述代码中[①]，我只是对 SMS 进行了解析并对其内容做了清理，获得了较为简洁的 SMS 文本。在接下来的代码中，我将会创建两个列表，分别用以获取被清理之后的所有 SMS 内容以及类标签。用 **ML（machine learning）**术语来说就是获取所有的 X 和 Y：

```
>>> smsdata = open('SMSSpamCollection') # check the structure of this file!
>>> smsdata_data = []
>>> sms_labels = []
>>> csv_reader = csv.reader(smsdata,delimiter='\t')
>>> for line in csv_reader:
>>>      # adding the sms_id
>>>     sms_labels.append( line[0])
>>>     # adding the cleaned text We are calling preprocessing method
```

① 译者注：此处的原文是 In the following code，但从文本实际布局来看，这里指的应该是这段文字上面的代码。疑为作者笔误。

```
>>>     smsdata.append(preprocessing(line[1]))
>>> sms.close()
```

在继续任何下一步动作之前，要确保自己所用的系统中已经安装了 scikit-learn 库。

```
>>> import sklearn
```



提示：

如果这句代码出了错，或者在安装 scikit 的过程中遇到了一些困难，可以按照下面链接中的内容来安装 scikit 库: http://scikit-learn.org/stable/install.html。

6.3　取样操作

一旦以列表的形式持有了整个语料库，接下来就要对其进行某种形式的取样操作。 通常来说，对语料库的整体取样方式与图 6-2 中的训练集、开发测试集和测试集的取样方式是类似的，整个练习背后的思路是要避免训练过度。如果将所有数据点都反馈给该模型，那么算法就会基于整个语料库来进行机器学习，但这些算法在真实测试中针对的是不可见数据。在非常简单的词汇环境中，如果在模型学习过程中使用的是全体数据，那么尽管分类器在该数据上能得到很好的执行，但其结果是不稳健的。原因在于一直只在给定数据上执行出最佳结果，但这样它是学不会如何处理未知数据的。

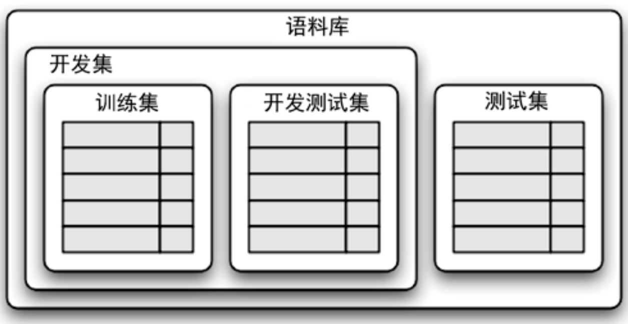


图 6-2

要想解决此类问题，最好的办法是将整个语料库划分成两个主要集合。在建模练习中，应该要避开开发集和测试集，只用开发测试集来完成建模操作。在完成整个建模练习之后，再将其结果放到之前搁置的测试集合中来进行预测。这样一来，如果该模型在该集合上表现良好，就可以确信它对任何新的数据样本都可以进行准确而稳健的预测。

取样本来就是一个非常复杂的操作流程，机器学习社区一直在对其深入研究，它本质上是一个应对许多数据编程和训练过度问题的补救措施。简单起见，本章将只进行基本取样，下面对语料库进行 70:30 的划分：

```
>>> trainset_size = int(round(len(sms_data)*0.70))
>>> # i chose this threshold for 70:30 train and test split.
>>> print 'The training set size for this classifier is ' + str(trainset_
size) + '\n'
>>> x_train = np.array([''.join(el) for el in sms_data[0:trainset_size]])
>>> y_train = np.array([el for el in sms_labels[0:trainset_size]])
>>> x_test = np.array([''.join(el) for el in sms_data[trainset_
size+1:len(sms_data)]])
>>> y_test = np.array([el for el in sms_labels[trainset_size+1:len(sms_
labels)]])or el in sms_labels[trainset_size+1:len(sms_labels)]])
>>> print x_train
>>> print y_train
```

- 如果将全体数据都用作训练数据，你认为情况会怎样？
- 如果面对的是一个非常不平衡的样本，情况又会怎样？

> **提示：**
> 如果想要了解更多可用的取样技术，请访问以下链接：
> http://scikit-learn.org/stable/modules/classes.html#module-sklearn.cross_validation。

下面将视线转到另一件事上：就是要将整个文本转换成向量形式。这种形式被称之为**词汇文档矩阵（term-document matrix）**。如果有必要为这个给定例子构建一个词汇文档矩阵，它看起来应该像下面这样：

TDM	anymore	call	camera	color	cried	enough	entitled	free	gon	had	latest	mobile
SMS1	0	1	1	1	0	0	1	2	0	1	0	3
SMS2	1	0	0	0	1	1	0	0	1	0	0	0

当然，文本文档也可以用所谓的 **BOW（bag of word）**来表示，这也是文本挖掘和其他相关应用中最常见的表示方法之一。基本上，不必去考虑这些单词在相关语境下的表示方式。

如果想要用 Python 来生成一个类似词汇文档矩阵，就需要用到 scikit 中的向量化器（vectorizer）：

```
>>>from sklearn.feature_extraction.text import CountVectorizer
>>>sms_exp=[ ]
>>>for line in sms_list:
>>>    sms_exp.append(preprocessing(line[1]))
>>>vectorizer = CountVectorizer(min_df=1)
>>>X_exp = vectorizer.fit_transform(sms_exp)
>>>print "||".join(vectorizer.get_feature_names())
>>>print X_exp.toarray()
array([[ 1, 0, 1, 1, 1, 0, 0, 1, 2, 0,
1, 0, 1, 3, 1, 0, 0, 0, 1, 0, 0, 2,
0, 0], [ 0, 1, 0, 0, 0, 1, 1, 0, 0, 1,
0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0,
1, 1, ]])
```

计数向量开了个好头，但它在使用过程中会遇到一个问题：即较长文档所获得的平均计数值会高于较短的文档，即使在讨论主题相同的时也是如此。

> **小技巧：**
> 如果想要避免这些潜在的误差，只要用文档中每个单词出现的次数除以该文档中的单词总数就行了。这个新的特征值叫做 tf（term frequencies）。

tf 之上还有另一个更细致的改进，那就是对语料库中许多文档中出现的词汇进行降格加权。通过这种方式，就可以得到减少那些只在该语料库的某一小部分中出现的信息。

这种降格加权被称为 tf-idf（**term frequency–inverse document frequency**）。幸运的是，scikit 库也提供了相应的实现方式，具体如下：

```
>>>from sklearn.feature_extraction.text import TfidfVectorizer
>>>vectorizer = TfidfVectorizer(min_df=2, ngram_range=(1, 2), stop_
words='english', strip_accents='unicode', norm='l2')
>>>X_train = vectorizer.fit_transform(x_train)
>>>X_test = vectorizer.transform(x_test)
```

现在得到了一个矩阵格式的文本，它与任何机器学习作业中得到的结果是一样的。现在，X_train 和 X_test 可以被用于所有机器学习算法中的分类处理了。所以接下来就要看看在文本分类这个语境中，有哪些最常用的机器学习算法。

6.3.1 朴素贝叶斯法

下面就来构建第一个文本分类器吧。首先来看朴素贝叶斯分类器。朴素贝叶斯分类器

依赖于贝叶斯算法，它本质上是一个根据给定特征/属性，基于某种条件概率为样本赋予某个类别标签的模型。在这里，将用频率/伯努利数来预估先验概率和后验概率。

$$后验概率=\frac{先验概率\times似然函数}{证据因子}$$

朴素算法往往会假设其中所有的特征都是相互独立的，这样对于文本环境来说看起来会直观一些。但令人惊讶的是，朴素贝叶斯算法在大多数实际用例中的表现也相当良好。

朴素贝叶斯（NB）法的另一个伟大之处在于它非常简单，实现起来很容易，评分也很简单。只需要将各频率值存储起来，并计算出概率。无论在训练时还是测试（评分）时，它的速度都很快。基于以上原因，大多数的文本分类问题都会用它来做基准。

下面就来写一下这个分类器的实现代码：

```
>>>from sklearn.naive_bayes import MultinomialNB
>>>clf = MultinomialNB().fit(X_train, y_train)
>>>y_nb_predicted = clf.predict(X_test)
>>>print y_nb_predicted
>>>print ' \n confusion_matrix \n '
>>>cm = confusion_matrix(y_test, y_pred)
>>>print cm
>>>print '\n Here is the classification report:'
>>>print classification_report(y_test, y_nb_predicted)
confusion_matrix [[1205 5]
                  [26 156]]
```

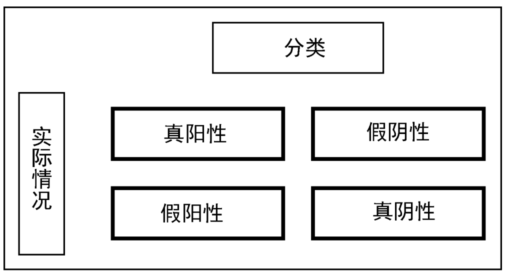


图 6-3

该方法将测试集中的所有 1 392 个样本读取到了混合矩阵中，其中有 1 205 个真阳性判例和 156 个真阴性判例。同时也预测到了 5 个假阴性判例和 26 个假阳性判例。其分类如图 6-3 所示。现在，对于这样一个典型的二元分类器，有不同的测量指标。

下面，就来给出其中几个最常见的分类测量指标的定义：

$$\text{Accuracy}=\frac{tp+tn}{tp+tn+fp+fn}$$

$$\text{Precision}=\frac{tp}{tp+fp}$$

$$\text{Recall}=\frac{tp}{tp+fn}$$

$$F=2\cdot\frac{\text{precision}\cdot\text{recall}}{\text{precision+recall}}$$

现在来看一下分类报告：

```
             Precision    recall     f1-score     support
ham          0.97         1.00       0.98         1210
spam         1.00         0.77       0.87         182
avg / total  0.97         0.97       0.97         1392
```

有了上述定义，现在结果一目了然。因此事实上，上述所有的测量指标看起来都挺不错的，这意味着分类器执行得准确而稳健。在这里，我会强烈建议你以更多的选项来查看该模块的测量指标，用来分析该分类器所得到的结果。这之中最重要、且最平衡的指标是 f1 指标（这是 R 关于精确率和反馈率的调和平均指标）。该指标之所以被广泛使用，是因为它给出了一个具有更高覆盖面的、优质的分类算法。另外，准确度也直观地告诉了我们真样本在所有样品中占了多少。精确率与反馈率也都有其各自的含义，精确率所讨论的是该分类器能得到多少真阳性判例以及它们的覆盖面，而反馈率则可以让我们详细了解自己能从这个关于真阳性和假阴性的判例池中得到多大的准确性。

提示：

如果想了解 scikit 库中各种类的更多信息，请访问以下链接:
http://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics。

下面来看另一个更重要的过程，读者要根据自己理解的模型真正深入地查看这个模型，通过查看实际特征来分辨阳性和阴性的判例类。在这里，只写了一段非常小的代码来生成前 n 个特征并打印出它们。具体如下：

```
>>>feature_names = vectorizer.get_feature_names()
>>>coefs = clf.coef_
>>>intercept = clf.intercept_
>>>coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))
>>>n = 10
>>>top = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
```

```
>>>for (coef_1, fn_1), (coef_2, fn_2) in top:
>>>    print('\t%.4f\t%-15s\t\t%.4f\t%-15s' % (coef_1, fn_1, coef_2,
fn_2))
-9.1602    10 den              -6.0396    free
-9.1602    15                  -6.3487    txt
-9.1602    1hr                 -6.5067    text
-9.1602    1st ur              -6.5393    claim
-9.1602    2go                 -6.5681    reply
-9.1602    2marrow             -6.5808    mobile
-9.1602    2morrow             -6.5858    stop
-9.1602    2mrw                -6.6124    ur
-9.1602    2nd innings         -6.6245    prize
-9.1602    2nd ur              -6.7856    www
```

上述代码只是从向量化器中读取了所有的特征名称，并获取了与给定特征相关的系数，然后将其中前 10 个特征打印出来。如果想要打印更多的特征，只需要修改代码中的 n 值即可。如果仔细观察这些特征，就会得到很多关于该模型的信息，以及更多与特征选择和其他参数相关的建议，如说关于预处理过程、单元/二元语法、词干提取、标记化处理等方面的建议。举个例子，查看垃圾过滤器的前几个特征，可以看到 2morrow、2nd innings 和一些非常明显的数字。对于阳性判例类（即垃圾信息）来说，“free”是一个非常明显被突出的词汇，许多垃圾信息里都有与免费优惠与交易相关的内容。当然，prize、www、claim 等其他词汇也同样值得关注的。

> **提示：**
> 更多细节请参考 http://scikitlearn.org/stable/modules/naive_bayes.html。

6.3.2 决策树

决策树是最古老的预测建模技术之一，对于给定的特征和目标，基于该技术的算法会尝试构建一个相应的逻辑树。使用决策树的算法有很多种类，这里主要介绍的是其中最着名和使用最广泛的算法之一：**CART**。

CART 算法会利用特性来构造一些二叉树结构，并构造出一个阈值，用于从每个节点中产生大量的信息。下面就通过编写代码来获取一个 CART 分类器：

```
>>> from sklearn import tree
>>> clf = tree.DecisionTreeClassifier().fit(X_train.toarray(), y_train)
>>> y_tree_predicted = clf.predict(X_test.toarray())
```

```
>>> print y_tree_predicted
>>> print ' \n Here is the classification report:'
>>> print classification_report(y_test, y_tree_predicted)
```

这里唯一的区别在于训练数据集的输入格式。需要将之前的稀疏矩阵格式修改成 **NumPy** 数组，因为 scikit 库树模块只接受一个 NumPy 数组。

通常情况下，只有在特征数量非常少时，树结构才是一个不错的选择。因此，尽管乍看之下我们在这里得到了不错的结果，但实际上人们很少会在文本分类问题上使用树结构。但从另一方面来说，树结构也确实有一些积极面。它仍然是一个最直观的算法，简单易懂，易于实现。基于树结构来实现的分类算法也很多，如 ID3、C4.5 和 C5 等。scikit-learn 库所采用的是 CART 算法的优化版本。

6.3.3 随机梯度下降法

随机梯度下降（Stochastic gradient descent，简称 SGD）法是一种既简单又非常有效的、适用于线性模型的方法。尤其在目标样本数量（和特征数量）非常庞大时，其作用会特别突出。如果参照之前的功能列表图，我们会发现 SGD 是许多文本分类问题的一站式解决方案。另外，由于它也能照顾到规范化问题并可以提供不同的损失函数，所以对于线性模型的实验工作来说它也是个很好的选择。

SGD 算法有时候也被称为**最大熵（Maximum entropy，简称 MaxEnt）算法**，它会用不同的（坡面）损失函数（loss function）和惩罚机制来适配针对分类问题与回归问题的线性模型。例如当 loss = log 时，它适配的是一个对数回归模型，而当 loss = hinge 时，它适配的则是一个线性的支持向量机（SVM）。

下面来看一个具体的 SGD 算法用例：

```
>>>from sklearn.linear_model import SGDClassifier
>>>from sklearn.metrics import confusion_matrix
>>>clf = SGDClassifier(alpha=.0001, n_iter=50).fit(X_train, y_train)
>>>y_pred = clf.predict(X_test)
>>>print '\n Here is the classification report:'
>>>print classification_report(y_test, y_pred)
>>>print ' \n confusion_matrix \n '
>>>cm = confusion_matrix(y_test, y_pred)
>>>print cm
```

其分类结果报告如下：

```
             precision   recall   f1-score   support
```

```
ham          0.99       1.00      0.99       1210
spam         0.96       0.91      0.93       182
avg / total  0.98       0.98      0.98       1392
```

下面是其最翔实的特征列表：

```
-1.0002    sir                    2.3815    ringtoneking
-0.5239    bed                    2.0481    filthy
-0.4763    said                   1.8576    service
-0.4763    happy                  1.7623    story
-0.4763    might                  1.6671    txt
-0.4287    added                  1.5242    new
-0.4287    list                   1.4765    ringtone
-0.4287    morning                1.3813    reply
-0.4287    always                 1.3337    message
-0.4287    and                    1.2860    call
-0.4287    plz                    1.2384    chat
-0.3810    people                 1.1908    text
-0.3810    actually               1.1908    real
-0.3810    urgnt                  1.1431    video
```

6.3.4 逻辑回归

逻辑回归（logistic regression）是一种针对分类问题的线性模型。它在某些文献中也被称为对元逻辑（logit regression）、最大熵（MaxEnt）分类法或对数线性分类器。在这个模型中，我们会用一个对元函数来进行建模，以概率的方式来描述单项试验的可能结果。

作为优化问题，L2 二元类所惩罚的逻辑回归可以用以下成本函数来进行最小化：

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log\left(\exp\left(-y_i\left(X_i^T w + c\right)\right) + 1\right)$$

类似地，由 L1 二元类所规范的逻辑回归则用以下函数来解决其优化问题：

$$\min_{w,c} \frac{1}{2} \|w\|_1 + C \sum_{i=1}^{n} \log\left(\exp\left(-y_i\left(X_i^T w + c\right)\right) + 1\right)$$

6.3.5 支持向量机

支持向量机（Support vector machine，简称 SVM）是目前在机器学习领域中最为先进的算法。

SVM 属于非概率分类器。SVM 会在无限维空间中构造出一组超平面，它可被应用在分类、回归或其他任务中。直观来说，可以通过一个超平面来实现良好的分类划界，这个超平面应该距离最接近训练数据点的那些类最远（这个距离被称为功能边界），因为在一般情况下，这个边界越大，分类器的规模就越小。

下面就用 scikit 库来构建一个最高级的监督式学习算法：

```
>>>from sklearn.svm import LinearSVC
>>>svm_classifier = LinearSVC().fit(X_train, y_train)
>>>y_svm_predicted = svm_classifier.predict(X_test)
>>>print '\n Here is the classification report:'
>>>print classification_report(y_test, y_svm_predicted)
>>>cm = confusion_matrix(y_test, y_pred)
>>>print cm
```

其分类结果报告与之前相同：

```
              precision    recall  f1-score   support
         ham       0.99      1.00      0.99      1210
        spam       0.97      0.90      0.93       182
avg / total        0.98      0.98      0.98      1392

confusion_matrix  [[1204    6] [  17  165]]
```

下面是其最翔实的特征列表：

```
    -0.9657    road                    2.3724    txt
    -0.7493    mail                    2.0720    claim
    -0.6701    morning                 2.0451    service
    -0.6691    home                    2.0008    uk
    -0.6191    executive               1.7909    150p
    -0.5984    said                    1.7374    www
    -0.5978    lol                     1.6997    mobile
    -0.5876    kate                    1.6736    50
    -0.5754    got                     1.5882    ringtone
    -0.5642    darlin                  1.5629    video
    -0.5613    fullonsms               1.4816    tone
    -0.5613    fullonsms com           1.4237    prize
```

这些结果绝对是目前为止所试过的所有监督算法中最好的。介绍完这个算法之后，对监督式分类器的介绍也就到此结束了。目前有数以百万计的图书在介绍各种不同的机器学习算法，即使是针对个别特定算法，可供选择的书也有很多。但强烈建议读者务必要在对上述算法有了一个深入了解之后，然后再在实际应用中使用它们。

6.4 随机森林算法

随机森林是一种以不同决策树组合为基础来进行评估的合成型分类器。事实上，它比较适合用于在各种数据集的子样本上构建多决策树型的分类器。另外，该森林中的每个树结构都建立在一个随机的最佳特征子集上。最后，启用这些树结构的动作也找出了所有随机特征子集中的最佳子集。总而言之，随机森林是当前众多分类算法中表现最佳的算法之一。

下面来看一个具体的随机森林算法用例：

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> RF_clf = RandomForestClassifier(n_estimators=10)
>>> predicted = RF_clf.predict(X_test)
>>> print '\n Here is the classification report:'
>>> print classification_report(y_test, predicted)
>>> cm = confusion_matrix(y_test, y_pred)
>>> print cm
```

提示：

对于仍然希望用 NLTK 库来解决文本分类问题的读者，推荐你阅读下面链接中的内容：http://www.nltk.org/howto/classify.html。

6.5 文本聚类

与文本相关的另一个问题族系是无监督式分类问题。关于这类问题，最常见的一种问题描述是“我手里有数以百万计的（非结构化数据）文档，是否能找到一种方式将它们分组，以便赋予其有意义的类别？”到目前为止，只要掌握了被标注的数据样本，就可以构建一个相应的监督式算法，这些之前已经讨论过了。在这里，需要使用无监督的方式来对这些文本文档进行分组。

文本聚类法（有时也叫聚类法）是目前最为常见的无监督式分组方式之一。使用聚类法的算法有很多种选择。其中，我本人最常用的是 **k 均值法（k-means）**或**层次聚类法（hierarchical clustering）**。下面，就分别来看看它们是如何与语料库搭配使用的。

K 均值法

该方法非常直观，从其名称就可以看出它需要试着找出 k 组围绕着若干数据点的平均值。

因此，该算法首先要随机拾取一些数据点来充当所有数据点的中心。接下来，该算法会将所有数据点各自分配给离其最近的那个中心。在这过程中，每完成一次迭代，其中心就要重新计算一次，然后继续迭代，直到达到中心不再变化的状态（即达到算法饱和）。

该算法还有一种变体，这种变体可以用迷你块（mini batches）的方式来减少计算时间，同时还会试图优化相同的目标函数。

> **提示：**
> 这里说的迷你块（mini batches）指的是从输入数据中随机采样的子集。这些选项通常应该在目标数据集确实很大，希望减少训练时间时被纳入考虑。

下面来看看 k 均值法的具体用例：

```
>>> from sklearn.cluster import KMeans, MiniBatchKMeans
>>> true_k=5
>>> km = KMeans(n_clusters=true_k, init='k-means++', max_iter=100, n_ init=1)
>>> kmini = MiniBatchKMeans(n_clusters=true_k, init='k-means++', n_init=1,
init_size=1000, batch_size=1000, verbose=opts.verbose)
>>> # we are using the same test,train data in TFIDF form as we did in text
classification
>>> km_model=km.fit(X_train)
>>> kmini_model=kmini.fit(X_train)
>>> print "For K-mean clustering "
>>> clustering = collections.defaultdict(list)
>>> for idx, label in enumerate(km_model.labels_):
>>>         clustering[label].append(idx)
>>> print "For K-mean Mini batch clustering "
>>> clustering = collections.defaultdict(list)
>>> for idx, label in enumerate(kmini_model.labels_):
>>>         clustering[label].append(idx)
```

在上面的代码中，导入了 scikit-learn 库的 kmeans 和 minibatchkmeans，并使用了一直在运行用例所采用的相同训练数据。另外还用最后三行代码打印出了一个针对各个样本的聚类。

6.6　文本中的主题建模

在文本语料库的语境中，另一个焦点问题就是要找出给定文档的主题。主题建模这个

概念可以用许多不同的方式来解决。常用来对文本文档进行主题建模的方法主要有 **LDA（Latent Dirichlet allocation，即隐式狄利客雷分布）**和 **LSI（Latent semantic indexing，即潜在语义索引）**这两种。

在大多数行业中，通常都会有大量的无标签文本文档。可以在无标签语料库中获取到该语料库的初始状况，主题模型是一个很棒的选项，因为它不仅能给出相关的主题，还能对整个语料库进行分门别类，并将其主题数量传递给算法。。

在这里，会用一个叫做“gensim”的 Python 库来实现这些算法。所以先将注意转到如何在相同的 SMS 数据集上实现 LDA 和 LSI 的问题上来。从目前来看，问题的唯一变化就是要在该 SMS 数据中对不同的主题进行建模，同时希望了解这些文件各自属于的主题。在这方面，Wikipedia 的整个转储数据就是一个比较合适和现实的用例，可以在上面找到各种不同的、已经经过讨论的主题，还有来自客户的数十亿条评论/投诉，可以从中获得人们对于相关主题的讨论状况。

安装 gensim

安装 gensim 最简单的一种方法就是使用包管理器：

```
>>> easy_install -U gensim
```

当然，也可这样安装：

```
>>> pip install gensim
```

安装完成之后，就可以执行以下命令了：

```
>>> import gensim
```

> **提示：**
> 如果这过程中出现了任何错误，请参考以下链接内容：
> https://radimrehurek.com/gensim/install.html。

下面，来看一段代码：

```
>>> from gensim import corpora, models, similarities
>>> from itertools import chain
>>> import nltk
>>> from nltk.corpus import stopwords
>>> from operator import itemgetter
>>> import re
>>> documents = [document for document in sms_data]
>>> stoplist = stopwords.words('english')
```

```
>>> texts = [[word for word in document.lower().split() if word not in stoplist]
\ for document in documents]
```

如你所见，我们从 SMS 数据中读取了文档，同时还移除其中的停用词。虽然之前章节中的方法也可以做到这件事，但这里，要用特定的库来做它。

> **提示：**
> gensim 库中包含了所有典型的 NLP 功能，并提供了一些很棒的方法来创建各种不同的语料库格式，如 TFIDF、libsvm、market matrix 等。同时还提供了这些格式之间的转换方法。

在接下来的这段代码中，需要将文档列表转换为 BOW 模型，然后再将该模型转换为一个典型的 **TF-IDF** 语料库：

```
>>>dictionary = corpora.Dictionary(texts)
>>>corpus = [dictionary.doc2bow(text) for text in texts]
>>>tfidf = models.TfidfModel(corpus)
>>>corpus_tfidf = tfidf[corpus]
```

在有了所需格式的语料库之后，就可以用以下两种方法来给出主题的数量，该模型会试着用语料库字的所有文档构建出一个 LDA / LSI 模型：

```
>>>si = models.LsiModel(corpus_tfidf, id2word=dictionary, num_topics=100)
>>>#lsi.print_topics(20)
>>>n_topics = 5
>>>lda = models.LdaModel(corpus_tfidf, id2word=dictionary, num_topics=n_
topics)
```

一旦完成了这些建模工作，接下来就要去理解这些不同的主题了，了解各种不同的词汇各自代表的是什么主题。下面，打印出一份与前几大词汇相关的主题：

```
>>> for i in range(0, n_topics):
>>>         temp = lda.show_topic(i, 10)
>>>         terms = []
>>>         for term in temp:
>>>             terms.append(term[1])
>>>             print "Top 10 terms for topic #" + str(i) + ": "+ ", ".join(terms)
Top 10 terms for topic #0: week, coming, get, great, call, good, day, txt, like, wish
Top 10 terms for topic #1: call, ..., later, sorry, 'll, lor, home, min, free, meeting
Top 10 terms for topic #2: ..., n't, time, got, come, want, get, wat, need, anything
```

```
Top 10 terms for topic #3: get, tomorrow, way, call, pls, 're, send, pick, ..., text
Top 10 terms for topic #4: ..., good, going, day, know, love, call, yup, get, make
```

如果你仔细查看一下这些输出信息，就会看到其中有 5 个不同的主题，它们有着明显不同的含义。想像一下，如果是在 Wikipedia 或者其他 Web 页面的大型语料库上执行相同的练习，你就会知道这些语料库表达了哪些有意义的主题了。

6.7 参考资料

- http://scikit-learn.org/
- https://radimrehurek.com/gensim/
- https://en.wikipedia.org/wiki/Document_classification

6.8 小结

本章归根结底是在介绍文本挖掘技术的世界。本章希望能为你提供一份基本介绍，帮助你了解一些最常见的、用于解决文本分类/聚类问题的算法。我们知道这些概念将怎样帮助你构建出真正伟大的 NLP 应用（如垃圾过滤器、以域为中心的新闻订阅、网页分类等）。尽管本章的代码示例中没有用 NLTK 库来处理模块分类，但用 NLTK 库执行了所有的预处理步骤。强烈建议读者使用优于 NLTK 的 scikit-learn 库来处理所有的分类问题。本章还带你初步涉入了机器学习领域，以了解它可以解决的问题类型。讨论了机器学习技术在文本环境中的一些特定问题。除此之外，还讨论在文本的分类、聚类以及主题建模方面最常见的一些分类算法，并给出了充足的实现细节，以帮助读者完成相关的工作。当然即便如此，我也仍然认为读者需要针对这里的每个算法进行大量的阅读，了解它们的理论，以获得更深入的理解。

除上述内容外，还向读者介绍了整个处理流程，读者需要根据这个流程来处理所有文本挖掘问题相关的情况。这里涵盖了机器学习在实践方面的大部分内容，包括取样、预处理、建模以及模型评估等。

下一章内容不会直接涉及 NLTK/NLP，但会介绍一个很受数据科学家与 NLP 爱好者欢迎的工具。在大多数 NLP 问题中，都需要处理一些非结构化文本数据，Web 就是其中内容最为丰富、体量最大的数据源之一。下一章将会介绍如何从 Web 中收集数据，并有效地利用这些技术来构建一令人惊叹的 NLP 应用。

第 7 章 Web 爬虫

Web 无疑是当前最大的非结构化文本的存储库，如果知道如何对其内容进行爬取，就可以获得所有想要的数据。正因为如此，Web 爬取（web crawling）对于 NLTK 的爱好者来说是一项很值得学习的技术，而如何从 Web 中获取相关的数据正是本章所要介绍的主要内容。

本章将会用一个叫做 Scrapy 的神奇的 Python 库来编写一个 Web 爬虫。本章会为你详细介绍该库所有可用于配置的各种不同设置信息。还会编写一些最常见的蜘蛛策略以及多个与之相关的用例。另外，由于 Scrapy 库的使用需要我们对 Xpath、信息爬取，信息检索等与 Web 信息操作相关的概念有一个基本的了解。所以本章也会对这些主题进行一定的探讨，以确保读者在具体实现相关应用之前，能了解其在实践方面的相关知识。总而言之，在阅读完本章之后，希望读者能对 Web 爬虫有个更好的了解，并能掌握以下内容。

- 如何用 Scrapy 库编写出属于自己的爬虫。
- Scrapy 库的所有主要功能。

7.1 Web 爬虫

Google 无疑是当前最大的网页爬虫之一，它爬取的对象是整个万维网（WWW）。Google 必须对 Web 中现存的每一个页面进行遍历，并检索/爬取其所遍历到的全部内容。

Web 爬虫是一种系统性逐页浏览 Web 中的页面，并对其内容进行检索或爬取的计算机程序。而且，Web 爬虫还可以从已被爬取过的内容中解析出接下来要访问的 URL 集。因此，如果这些程序进程可以面对整个 Web 无限期地运行下去，是可以爬取到所有网页的。另外，

Web 爬虫也被叫作蜘蛛、机器人和检索器，它们只是同一事物的不同名称[1]。

在编写第一个爬虫程序之前，有那么几个要点需要先思考一下。以目前的技术来说，在用 Web 爬虫对网页进行遍历之前，应该要先决定需要选取什么类型的内容，要忽略的又是什么内容。例如对于搜索引擎这样的应用来说，通常应该要忽略掉所有的图像、js 文件、css 文件及其他非 HTML 文件，将注意力集中在那些可被索引并且可被搜索的 HTML 内容上。在某些信息提取引擎中，还需要选取特定的 HTML 标签或网页中特定的部分。另外，如果想要执行递归式爬取操作的话，还需要提取其中的 URL。这就进入了爬取策略这一话题中来。在这一话题中，需要决定递归策略是深度优先还是广度优先。可能会想要追踪下一网页上的所有 URL，那么只要采用深度优先策略来获取这些 URL 即可。也可能会想前往下一网页中的所有 URL，这样的话只需一路递归下去即可。

当然，还需要确保自己不会陷入自循环状态，因为基本上在大多数情况下，需要遍历的是某种图结构。为此，需要确保自己有一个清晰的应对页面重复访问的策略。其中，聚焦爬取（focused crawling）是一种最常被讨论的爬取策略。在该策略中，要知道自己在找什么域或主题，以及所要抓取的域。这其中的一些问题将会在蜘蛛这一节中做更为详细的讨论。

> **提示：**
> 推荐读者看看 Udacity 上的视频：
> https://www.youtube.com/watch?v=CDXOcvUNBaA。

7.2　编写第一个爬虫程序

从最基本的爬虫程序开始，这个爬虫将会被用来爬取某个 Web 页面上的全部内容。在这里，要用 Scrapy 来编写这个爬虫。Scrapy 库是 Python 语言环境下爬虫问题的最佳解决方案之一。本章将会讨论 Scrapy 库中各种不同的功能。因此先要安装一下 Scrapy。

可以用以下命令来安装。

请键入以下命令：

```
$ pip install scrapy
```

[1] 译者注：从后文看，作者似乎很喜欢将蜘蛛和爬虫这两个词混着用，所以提前说明一下倒也是个办法。

用包管理来安装 Scrapy 无疑是最简单的方式。下面来测试一下安装是否一切就绪。(理想情况下，Scrapy 现在应该已经被纳入到了 sys.path 变量中)：

```
>>> import scrapy
```

> **小技巧：**
> 如果在安装过程中遇到了任何错误，请参考：
> http://doc.scrapy.org/ en/latest/intro/install.html。

现在，Scrapy 库应该可以工作了。下面，就来看第一个蜘蛛应用示例吧：

```
$ scrapy startproject tutorial
```

在执行完上述命令之后，该示例就应该会呈现出如下目录结构：

```
tutorial/
    scrapy.cfg  #the project configuration file
    tutorial/      #the project's python module, you'll later import your
code from here.
        __init__.py
        items.py        #the project's items file.
        pipelines.py    #the project's pipelines file.
        settings.py     # the project's settings file.
        spiders/        #a directory where you'll later put your spiders.
              __init__.py
```

如你所见，该项目的顶层文件夹显示这个示例的名字是 tutorial。然后该目录中还有一个项目配置文件（scrapy.cfg)，该文件用于定义项目应使用何种设置文件，并指定该项目的部署 URL。除此之外，tutorial 项目中还有几个重要文件：setting.py 可以用来指定该项目将使用何种类型的目标管道（item pipeline）①和蜘蛛。接下来是 item.py 和 pipline.py，这两个文件定义的是在解析目标项时所需要的数据以及需要执行何种类型的预处理操作。最后，spider 文件夹中包含的是我们为若干特定 URL 所编写的不同蜘蛛。

对于本章首个测试性的蜘蛛，我们打算用它将一些新闻内容转储到某一本地文件中。为此需在/tutorial /spiders 路径下创建一个名为 NewsSpider.py 的文件。然后，就可以来编写第一个蜘蛛程序了：

```
>>>from scrapy.spider import BaseSpider
>>>class NewsSpider(BaseSpider):
>>>    name = "news"
```

---

① 译者注：在 Scrapy 中，目标项（items）是用来加载被爬取内容的容器，其结构上有点像 Python 中的字典类型，但额外提供了一些保护以减少错误。

```
>>>     allowed_domains = ["nytimes.com"]
>>>     start_URLss = [
>>>         'http://www.nytimes.com/'
>>>     ]
>>>def parse(self, response):
>>>         filename = response.URLs.split("/")[-2]
>>>         open(filename, 'wb').write(response.body)
```

在这个蜘蛛程序准备就绪之后，就可以使用以下命令开始进行爬取了：

```
$ scrapy crawl news
```

在执行完上述命令后，终端中应该会出现类似这样的日志信息：

```
[scrapy] INFO: Scrapy 0.24.5 started (bot: tutorial)
[scrapy] INFO: Optional features available: ssl, http11, boto
[scrapy] INFO: Overridden settings: {'NEWSPIDER_MODULE': 'tutorial. spiders',
'SPIDER_MODULES': ['tutorial.spiders'], 'BOT_NAME': 'tutorial'}
[scrapy] INFO: Enabled extensions: LogStats, TelnetConsole, CloseSpider,
WebService, CoreStats, SpiderState
```

如果没有看到像上面这样的日志信息，那么一定在之前做错了某些事。请检查一下该蜘蛛程序所在的位置以及其他与 Scrapy 相关的设置，如说与 crawl 命令匹配的蜘蛛名称是否正确，以及在 setting.py 文件中所配置的蜘蛛和目标项是否与实际情况一致。

只要一切顺利，就会在本地文件夹中看到一个名为 www.nytimes.com 的文件，其中包含了 www.nytimes.com 这一页面上的全部内容。

下面，来详细地看一下这个蜘蛛代码中所用到的一些关系词。

- name：这是 Scrapy 库为蜘蛛程序分配的标识符，便于它查找与该蜘蛛相应的 spider 类。因此，crawl 命令的参数应该始终与该名称相匹配。另外，还需要注意该名称是区分大小写的，必须要确保它的唯一性。
- start_urls：这是该蜘蛛程序将要爬取的 URL 列表。爬虫通常会以某个种子 URL 为起点开始爬取，通过对其调用 parse()方法来解析并查找下一个要爬取的 URL。当然在这里，我们可以提供一份可执行爬取动作的起点 URL 列表，而不只是一个种子 URL。
- parse()：通过调用该方法可以解析起点 URL 上的数据。其元素种类的逻辑将会由目标项的特定属性来选取。这样做可以简单地将 HTML 页面的整个内容转换为与许多可调用的解析方法一样复杂的、并可以针对各独立的目标项属性的不同选择器。

因此，这段代码只是指定了一个 URL（在这里就是 www.nytimes.com）以充当爬取的起点，

然后爬取了该页面上的全部内容。通常来说，爬虫程序要比这更复杂一些，要做的事会更多一些。现在先暂且退后一步，来了解一下这段代码背后究竟发生了哪些事。为此，先来看一下图 7-1。

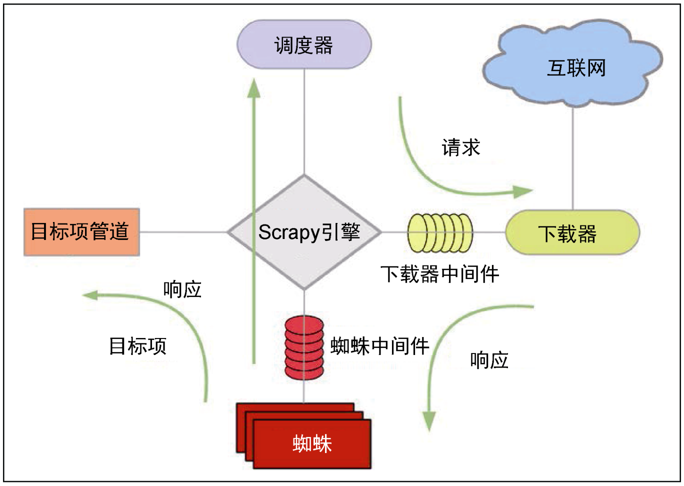


图 7-1

7.3　Scrapy 库中的数据流

在 Scrapy 中，数据流是由执行引擎所控制的，其主要流程如下所示。

1．该执行进程会找到之前所选择的蜘蛛程序并启动它，并打开 start_urls 列表中的第一个 URL。

2．接着，某个调度器会以请求的形式来调度这一个 URL。这更多的是属于 Scrapy 的内部操作。

3．然后，Scrapy 引擎会去继续查找下一组 URL 并对其进行爬取。

4．再接着，调度器会将下一批 URL 发送给执行引擎，而执行引擎会通过下载中间件将其转发给下载器。这些中间件位于存放不同代理和用户代理设置的地方。

5．然后，下载器会去下载来自相关页面的响应内容，并将其传递给蜘蛛程序，该蜘蛛的解析方法会从这段响应内容中选取特定的元素。

6．接下来，蜘蛛程序会将处理完的目标项发送给 Scrapy 引擎。

7．Scrapy 引擎再将处理完的响应内容发送给目标项管道，在该管道中还可以再添加一些后续处理。

8．Scrapy 引擎会继续对每个 URL 执行相同的过程，直到剩下的请求全部完成为止。

7.3.1　Scrapy 库的 shell

理解 Scrapy 库的最好方法就是在 shell 中使用它，亲自去使用一些由 Scrapy 库所提供的初始命令和工具。这些命令允许用户在练习和开发的过程中使用 XPath 表达式，可以将其放入自己的蜘蛛代码。

> **小技巧：**
> 想要在 Scrapy 库的 shell 中进行作业的话，建议你安装一下 Chrome 浏览器的开发插件和 Firebug（Mozilla Firefox 的插件）。这类工具对于从网页特定部分挖掘相关信息会很有帮助。

下面先来看一个非常有趣的用例。在这个用例中，任务是要从 Google 新闻（https://news.google.com/）中抓取出热门话题。

其具体步骤如下所示。

1．在喜欢的浏览器中打开 https://news.google.com/。

2．接下来，请切换到转到 Google 新闻的热门话题部分。并在第一个热门话题上单击右键，然后选择 Inspect Element 菜单项，其屏幕截图如图 7-2 所示。

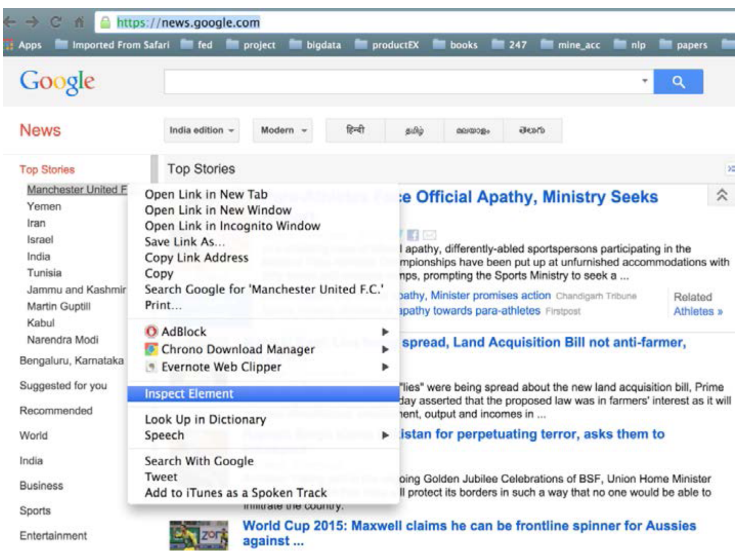


图 7-2

3．在打开该菜单项的一刻，浏览器会弹开一个侧边窗口，出现一个视图。

4．接着，请搜索并选取相关的 div 标签。如在这个例子中，我们感兴趣的是<div class = “topic” >。

5．待上述操作完成之后，就会了解到实际上自己已经完成了对相关网页的特定部分进行解析，其屏幕截图如图 7-3 所示。

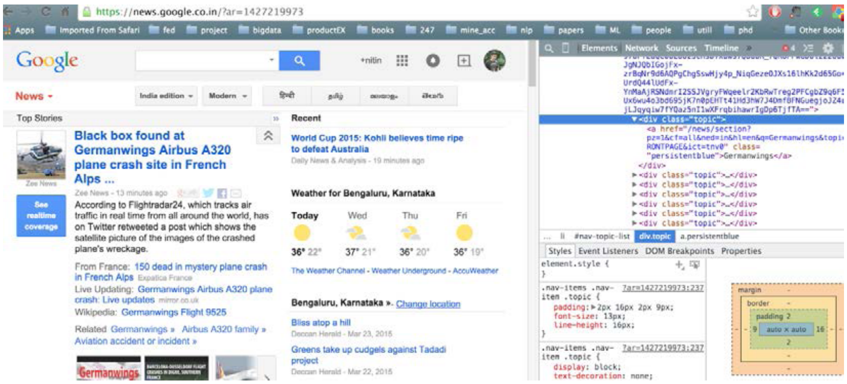


图 7-3

现在，事实上是要用自动化的方式来完成上面这些手动的操作步骤。Scrapy 库使用了一种名为 XPath 的 XML 路径语言。而 Xpath 是可以实现上面这类功能的。因此，下面就来看看 Scrapy 库是如何实现同一个例子的。

想要使用 Scrapy 库，需要先在命令行环境中输入以下命令：

```
$scrapy shell https://news.google.com/
```

在按下回车键的那一刹那，Google 新闻所在网页的响应内容就会被加载到 Scrapy 库的 shell 中。下面，将注意力转到 Scrapy 库中最重要的方面上来，了解如何查找网页中特定的 HTML 元素。现在，启动并运行上图中这个从 Google 新闻中获取相关话题的例子吧：

```
In [1]: sel.xpath('//div[@class="topic"]').extract()
```

然后，得到如下输出：

```
Out[1]:
[<Selector xpath='//div[@class="topic"]' data=u'<div class="topic"><a
href="/news/sectio'>,
<Selector xpath='//div[@class="topic"]' data=u'<div class="topic"><a
href="/news/sectio'>,
<Selector xpath='//div[@class="topic"]' data=u'<div class="topic"><a
```

```
href="/news/sectio'>]
```

现在，需要来了解一下在 shell 中用过的这些由 Scrapy 和 XPath 提供的函数。然后再继续更新蜘蛛程序，让它来执行一些更为复杂的工作。由于 Scrapy 选择器是在 lxml 库的辅助下构建的，这说明它们在速度和解析精度方面会非常相似。

下面来看一些最常用的选择器方法。

- xpath()：该方法会返回一个选择器列表，其中的每个选择器都代表了一个由 XPath 表达式参数所选中的节点。
- css()：该方法会返回一个选择器列表，其中的每个选择器都代表了一个由 CSS 表达式参数所选中的节点。
- extract()：该方法会以字符串的形式返回被选中数据的内容。
- re()：该方法会返回一个 unicode 字符串的列表，其内容是由给定的正则表达式参数所提取的。

稍后，会提供给你一份前 10 大选择器的清单，这些选择器足以覆盖到日常会涉及的大部分工作了。而对于更复杂的选择器，如果你在 Web 上搜索一番，就应该可以找到一个简单易用的解决方案了。下面，试着从网页中提取它的标题，这对于所有的网页都是很常见的任务：

```
In [2] :sel.xpath('//title/text()')
Out[2]: [<Selector xpath='//title/text()' data=u' Google News'>]
```

现在，在选取完元素之后，就会想继续执行更多的提取处理。下面就来提取被选取元素的内容。这是一个所有选择器都适用的通用方法：

```
In [3]: sel.xpath('//title/text()').extract()
Out[3]: [u' Google News']
```

除此之外，查看给定网页中的所有元素也是一个非常通用的请求操作。下面就用这个选择器来实现它：

```
In [4]: sel.xpath('//ul/li')
Out [4] : list of elements (divs and all)
```

也可以用下面这个选择器来提取网页中所有的标题：

```
In [5]: sel.xpath('//ul/li/a/text()').extract()
Out [5]: [ u'India',
u'World',
```

```
u'Business',
u'Technology',
u'Entertainment',
u'More Top Stories']
```

通过下面这个选择器，可以提取网页中所有的超链接：

```
In [6]:sel.xpath('//ul/li/a/@href').extract()
Out [6] : List of urls
```

下面要选取的是所有的<td>和 div 元素：

```
In [7]:sel.xpath('td'')
In [8]:divs=sel.xpath("//div")
```

接下来要选取的是所有的 div 元素，在这里可以使用循环：

```
In [9]: for d in divs:
    printd.extract()
```

上述代码会打印出整个网页中各个 div 元素中的全部内容。因此，在无法获取 div 的准确名称的情况下，也可以通过基于正则表达式的搜索功能来进行查看。

现在，再来选取所有包含属性 class =“topic”的 div 元素：

```
In [10]:sel.xpath('/div[@class="topic"]').extract()
In [11]:  sel.xpath("//h1").extract()          # this includes the h1 tag
```

下面来选取网页中所有的<p>元素，并获取这些元素的 class 属性信息：

```
In [12 ] for node in sel.xpath("//p"):
print node.xpath("@class").extract()
Out[12] print all the <p>
In [13]: sel.xpath("//li[contains(@class, 'topic')]")
Out[13]:
[<Selector xpath="//li[contains(@class, 'topic')]" data=u'<li class="navitem
nv-FRONTPAGE selecte'>,
<Selector xpath="//li[contains(@class, 'topic')]" data=u'<li
class="navitem nv-FRONTPAGE selecte'>]
```

接下来，要来编写一些用于从 css 文件中获取数据的选择器。如果只想从 css 文件中提取相关的标题，所需做的操作通常与之前相同，无非就是要做些语法上的修改：

```
In [14] :sel.css('title::text').extract()
Out[14]: [u'Google News']
```

通过下面的命令可以将网页中使用的所有图片名称列出来：

```
In[15]: sel.xpath('//a[contains(@href, "image")]/img/@src').extract()
Out [15] : Will list all the images if the web developer has put the images
in /img/src
```

最后，来看一下基于正则表达式的选择器：

```
In [16 ]sel.xpath('//title').re('(\w+)')
Out[16]: [u'title', u'Google', u'News', u'title']
```

在某些情况下，移除掉命名空间可以帮助我们获得正确的模式。选择器中通常都会内置 remove_namespaces()函数，该函数会确保相关文档被完整地扫描到，同时也会移除其中所有的命名空间。当然在调用这个函数之前，应该要先确认一下其中的一些命名空间是否属于模式的一部分。下面来看看 remove_namespaces()函数是如何被调用的：

```
In [17] sel.remove_namespaces()
sel.xpath("//link")
```

现在，我们对选择器应该有了更多的了解。下面继续来修改之前构建的那个新闻蜘蛛程序：

```
>>> from scrapy.spider import BaseSpider
>>> class NewsSpider(BaseSpider):
>>>     name = "news"
>>>     allowed_domains = ["nytimes.com"]
>>>     start_URLss = [
>>>         'http://www.nytimes.com/'
>>>     ]
>>> def parse(self, response):
>>>     sel = Selector(response)
>>>         sites = sel.xpath('//ul/li')
>>>         for site in sites:
>>>             title = site.xpath('a/text()').extract()
>>>             link = site.xpath('a/@href').extract()
>>>             desc = site.xpath('text()').extract()
>>>             print title, link, desc
```

如你所见，这里主要修改了 parse()方法，这是蜘蛛程序的核心方法之一。现在，这个蜘蛛程序就对整个网页进行了爬取操作，而且也对其中的标题、说明和 URL 进行了更结构化的解析。

现在，可以用 Scrapy 库中的所有功能来编写一个更健壮的爬虫程序了。

7.3.2　目标项

到目前为止，我们都只是将爬取所得的内容打印在标准输出上或转储到文件中。其实还有一个更好的选择：就是在每次编写爬虫程序的时候为其定义一个 items.py。这样做的好处是可以在自己的解析方法内使用这些目标项，并以任意的数据格式来输出，如 XML、JSON 或 CSV 等。所以，下面回到之前的爬虫程序中为其添加一个目标项类，其功能如下：

```
>>> fromscrapy.item import Item, Field
>>> class NewsItem(scrapy.Item):
>>>     # define the fields for your item here like:
>>>     # name = scrapy.Field()
>>>     pass
```

接下来，为该类添加如下三个不同的字段：

```
>>> from scrapy.item import Item, Field
>>> class NewsItem(Item):
>>>     title = Field()
>>>     link = Field()
>>>     desc = Field()
```

如你所见，通过 field()添加了 title、link 和 desc 这三个字段。在放置完字段后，上述蜘蛛程序的解析方法就可以被修改为 parse_news_item()了，因为它不会再将被解析的字段转储到某个文件中了，它现在使用的是目标项对象。

接下来要定义的是 Rule()方法，这是一种用于指定后续要爬取何种 URL 的方式。Rule()方法提供了 SgmlLinkExtractor()，这是一种用于定义 URL 模式的方式，它决定了能从被爬取网页中提取出哪些 URL。除此之外，Rule()方法还提供了一个回调方法，该方法通常会返回一个指针，蜘蛛程序会根据它来查找相关的解析方法，在这里就是 parse_news_item()。如果目标可以有不同的解析方式，那么也可以设置多个规则和解析方法。另外，Rule()方法还有一个布尔参数，可以用它来指定是否使用该规则去提取每个响应后面的链接。在回调函数为 None 的情况下，其默认值为 True，否则默认为 False。

在这里，需要重点注意的是 Rule()方法本身不是用来解析的。因为其默认回调的方法名是 parse()，如果要使用它，实际上就是要覆盖它的默认实现，并且可以以此来停止蜘蛛程序的爬取功能。接下来，将注意力转向下面的代码，来具体理解上面所提到的这些方法和参数：

```
>>> from scrapy.contrib.spiders import CrawlSpider, Rule
>>> from scrapy.contrib.linkextractors.sgml import SgmlLinkExtractor
>>> from scrapy.selector import Selector
>>> from scrapy.item import NewsItem
>>> class NewsSpider(CrawlSpider):
>>>     name = 'news'
>>>     allowed_domains = ['news.google.com']
>>>     start_urls = ['https://news.google.com']
>>>     rules = (
>>>         # Extract links matching cnn.com
>>>         Rule(SgmlLinkExtractor(allow=('cnn.com', ), deny=(http://
edition.cnn.com/', ))),
>>>         # Extract links matching 'news.google.com'
>>>         Rule(SgmlLinkExtractor(allow=('news.google.com', )),
callback='parse_news_item'),
>>>     )
>>>     def parse_news_item(self, response):
>>>         sel = Selector(response)
>>>         item = NewsItem()
>>>         item['title'] = sel.xpath('//title/text()').extract()
>>>         item[topic] = sel.xpath('/div[@class="topic"]').extract()
>>>         item['desc'] = sel.xpath('//td//text()').extract()
>>>         return item
```

7.4 生成网站地图的蜘蛛程序

如果相关网站提供了 sitemap.xml，那么使用 SiteMapSpider 来对该网站进行爬取无疑是一种更好的选择。

在给定 sitemap.xml 的情况下，蜘蛛程序所解析的是由网站自身提供的 URL。这无疑是一种更优雅的爬取方式，也会是一种更良好的实践：

```
>>> from scrapy.contrib.spiders import SitemapSpider
>>> class MySpider(SitemapSpider):
>>>     sitemap_URLss = ['http://www.example.com/sitemap.xml']
>>>     sitemap_rules = [('/electronics/', 'parse_electronics'),
('/ apparel/', 'parse_apparel'),]
>>>     def 'parse_electronics'(self, response):
>>>         # you need to create an item for electronics,
>>>         return
>>>     def 'parse_apparel'(self, response):
>>>         #you need to create an item for apparel
>>>         return
```

在上面这段代码中，我们为每个产品目录都编写了一个相应的解析方法。如果想建立一个价格汇总/比对程序的话，这显然是个不错的用例。在这里，需要针对不同产品的不同属性进行解析，例如对于电子类产品，可能要收集的是它的技术规格、附件以及价格信息；对于服装类产品，可能就更关心它们的大小和颜色。建议读者找一个零售类网站来亲自尝试一下，在 shell 中通过相关的获取模式来收集不同目标项的大小、颜色与价格信息。如果这样做了，就等于在一个良好的状态下写出了第一个符合工业标准的蜘蛛程序。

在某些情况下，想要爬取的网站必须要先行登录，然后才能进入该网站的某些部分。Scrapy 库目前对此也有了一个解决方法。它们实现了 FormRequest 对象，该对象将以 POST 的形式来呼叫 HTTP 服务器并获得响应。下面，就通过具体的蜘蛛程序代码来深入地了解一下：

```
>>> class LoginSpider(BaseSpider):
>>>     name = 'example.com'
>>>     start_URLss = ['http://www.example.com/users/login.php']
>>>     def parse(self, response):
>>>         return [FormRequest.from_response(response,
formdata={'username': 'john', 'password': 'secret'}, callback=self.after_
login)]
>>>     def after_login(self, response):
>>>         # check login succeed before going on
>>>         if "authentication failed" in response.body:
>>>             self.log("Login failed", level=log.ERROR)
>>>           return
```

对于只需要输入用户名和密码而无需任何验证码的网站，只需在上述代码中添加特定的详细登录信息即可。这也是解析方法的一部分，因为大多数情况下都想要在首页中进行登录的。在完成登录之后，就可以针对目标项以及其他细节信息来编写属于自己的回调方法 after_login()了。

7.5 目标项管道

下面，来谈一谈关于目标项的后续处理。在这方面，Scrapy 库提供了一种为目标项定义管道的方式，可以通过这种方式来定义相关的目标项后续要执行怎样的处理。这是一种有条不紊的、优秀的程序设计思维。

如果想要对之前所收集的目标项进行处理，如移除干扰词、执行大小写转换以及需要对目标中某个值进行其他处理，例如想根据出生日期（DOB）来计算年龄，或者根据原价

格来计算折扣，这些都需要专门构建属于自己的目标项管道。而且到最后，可能还需要通过这些管道将目标项各自转储到某个文件中。

下面就来看看这种方式的具体实现步骤。

1．首先，需要在 setting.py 中定义一个目标项管道：

```
ITEM_PIPELINES = {
    'myproject.pipeline.CleanPipeline': 300,
    'myproject.pipeline.AgePipeline': 500,
    'myproject.pipeline.DuplicatesPipeline: 700,
    'myproject.pipeline.JsonWriterPipeline': 800,
}
```

2．然后，来编写一个类，清理一下目标项：

```
>>>from scrapy.exceptions import Item
>>>import datetime
>>>import datetime
>>>class AgePipeline(object):
>>>    def process_item(self, item, spider):
>>>      if item['DOB']:
>>>          item['Age'] = (datetime.datetime.
strptime(item['DOB'], '%d-%m-%y').date()-datetime.datetime.
strptime('currentdate, '%d-%m-%y').date()).days/365
>>>          return item
```

3．接着，需要根据出生日期来推导出年龄，这里将会用到 Python 的日期函数：

```
>>>from scrapy import signals
>>>from scrapy.exceptions import Item
>>>class DuplicatesPipeline(object):
>>>    def __init__(self):
>>>        self.ids_seen = set()
>>>    def process_item(self, item, spider):
>>>        if item['id'] in self.ids_seen:
>>>           raise DropItem("Duplicate item found: %s" % item)
>>>        else:
>>>           self.ids_seen.add(item['id'])
>>>           return item
```

4．还需要移除掉重复的内容。由于 Python 的 set()数据结构可以确保其项目值的唯一性，所以我们可以用 Scrapy 库来创建一个管道，下面是其实现代码 DuplicatesPipeline.py：

```
>>> from scrapy import signals
```

```
>>> from scrapy.exceptions import Item
>>> class DuplicatesPipeline(object):
>>>     def __init__(self):
>>>         self.ids_seen = set()
>>>     def process_item(self, item, spider):
>>>         if item['id'] in self.ids_seen:
>>>             raise DropItem("Duplicate item found: %s" % item)
>>>         else:
>>>             self.ids_seen.add(item['id'])
>>>             return item
```

5．最后，要用 JsonWriterPipeline.py 中定义的管道将目标项写入到相关的 JSON 文件中：

```
>>>import json
>>>class JsonWriterPipeline(object):
>>>    def __init__(self):
>>>        self.file = open('items.txt', 'wb')
>>>    def process_item(self, item, spider):
>>>        line = json.dumps(dict(item)) + "\n"
>>>        self.file.write(line)
>>>        return item
```

7.6 参考资料

我们鼓励读者学习一些简单的蜘蛛程序实现，然后尝试着用它们来构建出一些很酷的应用程序。下面是本人所推荐的参考链接。

- http://doc.scrapy.org/en/latest/intro/tutorial.html。
- http://doc.scrapy.org/en/latest/intro/overview.html。

7.7 小结

在这一章中，我们学习了另一个非常优秀的 Python 程序库。从现在起，可以不必再依赖别人的数据了。因为已经学会了如何编写一个非常复杂的爬虫系统，以及如何编写一个专注于某类信息的蜘蛛程序。总之在本章，我们看到了从主系统中抽象出目标项逻辑的具体方法，以及如何为一些最常见的用例来编写特定的蜘蛛程序。在此过程中，了解了一些在实现自定义蜘蛛程序时最常用到的设置，也写了一些可重用的、复杂的解析方法。除此

之外，还对选择器进行了非常深入的了解，知道了如何用手动的方式来确定所需的、特定的目标项属性，也可以通过 Firebug 这样的工具来进一步实际地了解选择器。最后一个同样重要的建议是，请务必遵守你所爬取网站的安全指南。

下一章将会向你介绍一些基本的 Python 库在自然语言处理和机器学习领域的使用情况。

第 8 章
NLTK 与其他 Python 库的搭配运用

本章将会带你探索 Python 在机器学习和自然语言处理方面的一些主干库。到目前为止，前面已经使用过了 NLTK、Scikit 和 genism 这三个库，它们在功能上都非常抽象，所要处理的也都是非常具有针对性的任务。大多数统计型 NLP 都大量地依赖于向量空间模型，而向量空间模型的基础是线性代数的基本运算，这部分将由 NumPy 库所覆盖。除此之外，NLP 领域中有许多任务（如 POS 或 NER 标记）在卸下伪装之后，其实都是一些分类器。本章将会讨论所有这些任务中会被大量用到的程序库。

本章的主要用意是希望为读者提供一份最基本的 Python 库的快速预览。这将有助于读者了解更多这些最酷炫的程序库背后的数据结构、设计和数学，如之前章节中所讨论的 NLTK 和 Scikit。

下面是本章将要介绍的 4 个程序库。在这里，会尽量维持一份简介该有的篇幅，但如果你希望在数据科学领域掌握更多基于 Python 的一站式解决方案，我个人会强烈建议读者应该去阅读更多关于这些库的详细信息。

- NumPy（用于数值计算）。
- SciPy（用于科学计算）。
- pandas（用于数据操纵）。
- matplotlib（用于可视化处理）。

8.1 NumPy

NumPy 是一种用于处理数值计算的 Python 库，而且其运算速度很快。NumPy 库提供了一些高度优化的数据结构（如 ndarray）。另外，NumPy 库中也提供了许多为数值计算专

门设计和优化的函数，用于执行一些最常见的数值运算。因此，这个库也是 NLTK、scikitlearn、pandas 等其他相关库实现其一些算法的基础之一。本节会简单地介绍一些 NumPy 库的运行实例。这样做不仅有助于读者了解 NLTK 与其他相关库背后所用的基本数据结构，而且还能使读者有能力根据自己的需要自定义其中的一些功能。

下面先来讨论 ndarray，看看它们是如何被用作矩阵的，以及在 NumPy 中处理矩阵运算是何等简单、高效。

8.1.1 多维数组

ndarray 是一个数组对象，表示的元素类型单一、且元素数目固定的多维数组。

下面，先用一个普通的 Python 列表来构建 ndarray 对象：

```
>>> x=[1,2,5,7,3,11,14,25]
>>> import numpy as np
>>> np_arr=np.array(x)
>>> np_arr
```

如你所见，上面显示的是一个线性的单维数组。但 Numpy 的真正强大之处在于它的二维数组。接下来就来看看二维数组，用 Python 列表的列表来创建它。

```
>>> arr=[[1,2],[13,4],[33,78]]
>>> np_2darr= np.array(arr)
>>> type(np_2darr)
numpy.ndarray
```

索引操作

ndarray 的索引方式使其更像是一个 Python 容器。 NumPy 库可以通过切片的方式来提供对 ndarray 对象的不同观察方式。

```
>>> np_2darr.tolist()
[[1, 2], [13, 4], [33, 78]]
>>> np_2darr[:]
array([[1, 2], [13,  4], [33, 78]])
>>> np_2darr[:2]
array([[1, 2], [13, 4]])
>>> np_2darr[:1]
array([[1, 2]])
>>> np_2darr[2]
array([33, 78])
```

```
>>>    np_2darr[2][0]
>>> 33
>>>    np_2darr[:-1]
array([[1, 2], [13, 4]])
```

8.1.2　基本运算

NumPy 库还另外提供了一组可用于处理各种数值计算的操作。在下面这个例子中，希望以 0.1 的步长来获得一个包含 0 到 10 区间内所有数字的数组。对于任何优化例程来说，这都是一个典型需求。于是在一些最为常见的库（如 Scikit 和 NLTK）中，会用下面这些 NumPy 函数来处理问题。

```
>>> import numpy as np
>>> np.arange(0.0, 1.0, 0.1)
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9,  1]
```

在这里，既可以像上面这样做，也可以像下面这样生成一个元素全为 1 或 0 的数组：

```
>>> np.ones([2, 4])
array([[1., 1., 1., 1.], [1., 1., 1., 1.]])
>>> np.zeros([3,4])
array([[0., 0., 0., 0.], [0., 0., 0., 0.], [0., 0., 0., 0.]])
```

哇哦！

如果你当年做过高中数学，就会知道在许多代数运算中全都会涉及矩阵。你猜怎么着？大部分 Python 机器学习库也一样会用到它们！

```
>>>np.linspace(0, 2, 10)
array([   0.,    0.22222222,    0.44444444,    0.66666667,
0.88888889,    1.11111111,    1.33333333,    1.55555556,    1.77777778,
2,    ])
```

linspace()函数返回的是一组间隔相等的数字样本，它的计算范围位于设定的起始值和结束值之间。在上面这个给定示例中，想要获取的是 0 到 2 区间内的 10 个样本。

类似地，也可以用对数尺度来获取数组。其调用函数如下：

```
>>> np.logspace(0,1)
array([   1.,    1.04811313,    1.09854114,    1.1513954,    7.90604321,
8.28642773,    8.68511374,    9.10298178,    9.54095476,    10.,    ])
```

在这里，一样可以通过 Python 的 help()函数来获取相关参数和返回值的更多细节信息。

```
>>> help(np.logspace)
Help on function logspace in module NumPy.core.function_base:

logspace(start, stop, num=50, endpoint=True, base=10.0)
    Return numbers spaced evenly on a log scale.

    In linear space, the sequence starts at "base ** start"
    ('base' to the power of 'start') and ends with "base ** stop"
    (see 'endpoint' below).

    Parameters
    ---------
    start : float
```

如你所见，除了在调用时需提供起始值、结束值及想要获得的样本数，在上面这个用例中，还得提供一个基数。

8.1.3 从数组中提取数据

还可以在 ndarray 对象上执行各种数据操纵和过滤。下面，先来创建一个新的 Ndarray 对象——A。

```
>>> A = array([[0, 0, 0], [0, 1, 2], [0, 2, 4], [0, 3, 6]])

>>> B = np.array([n for n in range n for n in range(4)])
>>> B
array([0, 1, 2, 3])
```

现在，可以对其执行各类条件操作了，你将会在下面看到这些操作的示范，它们看起来都非常优雅：

```
>>>less_than_3 = B<3 # we are filtering the items that are less than 3.
>>>less_than_3
array([ True,  True,  True, False], dtype=bool)
>>>B[less_than_3]
array([0, 1, 2])
```

还可以将某个值赋予所有的这些值，如下所示。

```
>>> B[less_than_3] = 0
```

```
>>>: B
array([0, 0, 0, 3])
```

另外，还有一种用于获取指定矩阵对角线上数字的方法。下面是矩阵 A 对角线上的数字：

```
>>>np.diag(A)
array([0, 1, 4])
```

8.1.4 复杂矩阵运算

元素相乘是常见的一种矩阵运算，该运算需要将一个矩阵中的元素与另一个矩阵中的元素相乘。其结果应该是一个与输入矩阵形状相同的矩阵[1]，例如：

```
>>> A = np.array([[1,2],[3,4]])
>>> A * A
array([[ 1,  4], [ 9, 16]])
```

> **提示：**
>
> 但是不能执行下面这样的运算，它会在执行时抛出这样的错误信息：
>
> ```
> >>> A * B
> ---------------------------------------------
> ValueError Traceback (most recent call last)
> <ipython-input-53-e2f71f566704> in <module>()
> ----> 1 A*B
> ```
>
> **ValueError**：参与运算的对象在形状上(2,2) (4,)不一致。
>
> 简单地说，就是第一运算对象的列数必须要与第二运算对象的行数相匹配，这样才能执行矩阵的乘法运算。

下面再来看点积运算，该运算可是许多优化措施和代数运算的核心操作。我一直以来都觉得在传统环境下这件事做起来不是很有效率。现在来看看它在 NumPy 库中是多么容易，以及在内存方面是多么高效。

```
>>>np.dot(A, A)
array([[ 7, 10], [15, 22]])
```

当然，也可以执行像加、减和转置这样的操作，具体如下：

① 译者注：原文如此，其实矩阵乘法运算的结果举证在形状上应该是行数等于第一输入矩阵，列数等于第二输入矩阵。在这里是两个相同的矩阵相乘，自然在形状上也相同。

```
>>> A - A
array([[0, 0], [0, 0]])
>>> A + A
array([[2, 4], [6, 8]])
>>> np.transpose(A)
array([[1, 3], [2, 4]])
>>>> A
array([[1, 2], [2, 3]])
```

在这里，可以用下面这个操作来替代上面的转置运算：

```
>>> A.T
array([[1, 3], [2, 4]])
```

还可以先将这些 ndarray 对象转换为矩阵，再来执行矩阵运算，如下所示。

```
>>> M = np.matrix(A)
>>> M
matrix([[1, 2], [3, 4 ]])
>>> np.conjugate(M)
matrix([[1, 2], [3, 4]])
>>> np.invert(M)
matrix([[-2, -3], [-4, -5]])
```

可以用 NumPy 库来执行各种复杂的矩阵运算，而且使用起来也非常简单！这方面请读者自行查看有关 NumPy 的文档，以了解更详细的信息。

现在，回到一些常见的数学运算上来，比如找出给定数组中的最小值、最大值、平均值以及标准差。在下面的代码中，会生成一组正态分布的随机数，以便用它来示范这些运算的具体应用：

```
>>> N = np.random.randn(1,10)
>>> N
array([[    0.59238571,    -0.22224549,    0.6753678,    0.48092087,
-0.37402105,    -0.54067842,    0.11445297,    -0.02483442,
-0.83847935,    0.03480181,    ]])
>>> N.mean()
-0.010232957191371551
>>> N.std()
0.47295594072935421
```

这只是一个示例，目的是演示如何通过 NumPy 库来执行简单的数值计算和代数运算，

找出一组数字中的平均值和标准差。

1. 重塑与堆叠

在某些数值计算和代数运算中，可能会需要在输入矩阵的基础上改变结果矩阵的形状。在这方面，NumPy 库提供了一些简单有效的方式来重塑（reshaping）和堆叠（stacking）矩阵，以满足我们的任何期待。

```
>>> A
array([[1, 2], [3, 4]])
```

如果这里想要的是一个扁平矩阵，那么就只需要对其调用 NumPy 库中的 reshape()函数，将它重塑：

```
>>>> (r, c) = A.shape  # r is rows and c is columns
>>>> r,c
(2L, 2L)
>>>> A.reshape((1, r * c))
array([[1, 2, 3, 4]])
```

这种重塑操作在许多代数运算中都会用到。另外，如果只是想展平 ndarray 对象，也可以调用 flatten()函数：

```
>>>A.flatten()
array([1, 2, 3, 4])
```

另外，还可以通过一个函数给指定数组重复填充相同的元素。可以根据需要指定这些元素被重复的次数。例如想要对 ndarray 对象中的元素进行重复的话，可以这样调用 repeat()函数：

```
>>> np.repeat(A, 2)
array([1, 1, 2, 2, 3, 3, 4, 4])
>>> A
array([[1, 2],[3, 4]])
```

在上述例子中，每个元素按顺序被重复两次。另外，还有一个功能类似的 tile()函数也可以用于重复矩阵，具体使用如下：

```
>>> np.tile(A, 4)
array([[1, 2, 1, 2, 1, 2, 1, 2], [3, 4, 3, 4, 3, 4, 3, 4]])
```

除此之外，还可以为矩阵添加新的行或列。如，如果想添加一行，就可以调用

concatenate()函数：

```
>>> B = np.array([[5, 6]])
>>> np.concatenate((A, B), axis=0)
array([[1, 2], [3, 4], [5, 6]])
```

这一功能也可以通过调用 Vstack()函数来实现，例如：

```
>>> np.vstack((A, B))
array([[1, 2], [3, 4], [5, 6]])
```

当然，如果想要添加的是一列，也只需按以下方式来调用 concatenate()函数：

```
>>> np.concatenate((A, B.T), axis=1)
array([[1, 2, 5], [3, 4, 6]])
```

小技巧：
或者，你也可以通过 hstack()函数来添加列。这与之前调用 vstack()函数的方式非常相似。

2. 随机数

随机数的生成也是 NLP 和机器学习领域中许多任务都会涉及的操作。下面就来看看在这里获取随机样本是多么简单：

```
>>> from numpy import random
>>> #uniform random number from [0,1]
>>> random.rand(2, 5)
array([[ 0.82787406, 0.21619509, 0.24551583, 0.91357419, 0.39644969],
[ 0.91684427, 0.34859763, 0.87096617, 0.31916835, 0.09999382]])
```

除此之外，还有一个名为 random.randn()的函数也可以用来生成随机数，它会在给定区间内生成正态分布的随机数。如在下面的示例中，将会在 2 到 5 之间生成随机数。

```
>>> random.randn(2, 5)
array([[-0.59998393, -0.98022613, -0.52050449, 0.73075943, -0.62518516],
[ 1.00288355, -0.89613323,  0.59240039, -0.89803825, 0.11106479]])
```

这就是通过调用 random.randn(2,5)这个函数来实现的。

8.2 SciPy

这个科学计算版的 Python 或者说 SciPy 库是一个构建在 NumPy 库及其 ndarray 对象基础之上的框架，它基本上是一个为处理高级科学计算（如优化操作、积分运算、代数运算和傅里叶变换等）而开发的库。

该库的主要思路是在高效的内存管理器中，通过有效运用 ndarray 对象来提供常见的科学算法。因为 NumPy 和 SciPy 的存在，可以将自己的注意力集中在编写 scikit-learn 和 NLTK 这样的专用库上，处理一些专用领域的问题。在这方面，NumPy / SciPy 这样的库提供了很大的便利。下面，我们会简单地为你介绍一些由 SciPy 库所提供的数据结构和常见操作，还将会带你了解一些黑盒库的详细信息，例如会介绍 scikit-learn 库，带你了解一下其中的一些内幕。

```
>>> import scipy as sp
```

上面演示的是引入 SciPy 库的方式。我们在这里为它起了个别名，你可以按自己的意愿来进行引入。

下面，先从自己比较熟悉的东西开始，来看看如何通过调用 quad()函数来实现积分运算。

```
>>> from scipy.integrate import quad, dblquad, tplquad
>>> def f(x):
>>>     return x
>>> x_lower == 0 # the lower limit of x
>>> x_upper == 1 # the upper limit of x
>>> val, abserr = = quad(f, x_lower, x_upper)
>>> print val,abserr
>>> 0.5 , 5.55111512313e-15
```

如果求的是 x 的积分，那么它就应该是 x2/2，即 0.5。当然，库中还有其他用于科学计算的函数，具体如下所示。

- 插值运算（scipy.interpolate）。
- 傅里叶变换（scipy.fftpack）。
- 信号处理（scipy.signal）。

但在这里，会把焦点集中在线性代数和优化措施的议题上，因为它们与机器学习和 NLP

的关系更密切一些。

8.2.1 线性代数

SciPy 的线性代数模块中包含了大量与矩阵相关的函数。这个库对于业界的最大贡献，恐怕就是其对稀疏矩阵（CSR 矩阵）的支持了，很多其他涉及矩阵运算的库都会用到它。

SciPy 所提供的是当前业界用来存储稀疏矩阵以及对其进行相关数据操作的最佳方法之一。除此之外，它也提供了一些常见的操作，如线性方程的求解。它能很好地解决特征值和特征向量、矩阵函数（例如矩阵取幂运算）以及其他更复杂的操作（例如奇异值分解（SVD））。其中有一些操作是在机器学习历程中常会用到的幕后优化措施。例如，这里提到 SVD 正是第 6 章所介绍的 LDA（主题建模）的最简单形式。

下面，通过一个示例来演示一下线性代数模块的具体用法：

```
>>> A = = sp.rand(2, 2)
>>> B = = sp.rand(2, 2)
>>> import Scipy
>>> X = = solve(A, B)
>>> from Scipy import linalg as LA
>>> X = = LA.solve(A, B)
>>> LA.dot(A, B)
```

提示：
关于这方面更详细的信息，请参阅文档：
https://docs.scipy.org/doc/scipy/reference/linalg.html。

8.2.2 特征值与特征向量

在一些 NLP 和机器学习相关的应用中，会将文档表示成词汇文档矩阵的形式。通常会根据许多不同的数学公式来计算它们的特征值和特征向量。例如说 A 是矩阵，那么它就一定会存在一个向量 v，使得 $Av=\lambda v$。

在这里，λ 就是特征值，v 就是特征向量。另外，作为最常用的操作之一，奇异值分解（SVD）操作中会需要用到一些微积分功能。这在 SciPy 中是很容易实现的。

```
>>> evals = LA.eigvals(A)
>>> evals
array([-0.32153198+0.j, 1.40510412+0.j])
```

下面来获取它的特征向量：

```
>>> evals, evect = LA.eig(A)
```

还可以执行一些别的矩阵运算，如求逆、转置和行列式：

```
>>> LA.inv(A)
array([[-1.24454719, 1.97474827], [ 1.84807676, -1.15387236]])
>>> LA.det(A)
-0.4517859060209965
```

8.2.3 稀疏矩阵

在现实的应用场景中，通常会用到的都是一些大多数元素为 0 的矩阵。对于所有的矩阵运算来说，要忽略这些非 0 元素都是非常低效的。为了解决这类问题，引入了一种稀疏矩阵的格式，以便以一种简单的思路来存储那些非 0 元素。

简而言之，就是这种格式将大多数元素是非 0 的矩阵称为密集矩阵，而大多数元素为 0 的矩阵则被称为稀疏矩阵。

正如你所知道的，矩阵通常都是一些由行和列索引其元素值的二维数组。但现在来介绍几种不同的存储稀疏矩阵的方式。

- **DOK**（**键字典**）。在这种情况下，字典的键会被存储成（row, col）格式，而它的值就是所要存储的元素值。
- **LOL**（**列表的列表**）。在这种情况下，为目标结构的每一行设置一个列表，并且该列表只索引非 0 元素。
- **COL**（**坐标列表**）。在这种情况下，元素会以（row, col, value）列表的形式被存储成一个列表。
- **CRS / CSR**（**按行压缩存储**）。CSR 矩阵会先按行读取元素，然后用列索引的方式来存储每一个元素值，同时将相应行的指针存储下来，表示成（val，col_ind，row_ptr）三元组。这里的 val 就是一个包含矩阵中非 0 元素的数组，col_ind 则表示的是对应元素的列索引，最后的 row_ptr 则指向了 val 中被索引元素值所在的列表，它是各行的起始位置[①]。这种方式的名称反映了一个事实，就是它事实上是对 COO 存储格式进行了压缩。这种格式能有效地处理算术运算、列切片操作、矩阵向量的乘积等问题。

① 译者注：原文在这里似乎是将 CSR 描绘成了 CSC，然后又在后面的 CSC 重复了一遍，译者在这里做了自行更正。

> **提示：**
> 更详细的信息请参阅：
> http://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.sparse.csr_matrix.html。

- **CSC**（**稀疏列**）①。这种方式与 CSR 基本相同，只不过这里的值要先按列读取，然后用行索引存储每个元素，并存储列指针。换句话说，CSC 的表示形式是（val, row_ind, col_ptr）。

> **提示：**
> 这部分内容也可以查看文档：
> http://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.sparse.csc_matrix.html。

下面，就来亲自实践一下 CSR 矩阵的操作。假设现在有一个稀疏矩阵 ***A***：

```
>>> from scipy import sparse as s
>>> A = array([[1,0,0],[0,2,0],[0,0,3]])
>>> A
array([[1, 0, 0], [0, 2, 0], [0, 0, 3]])
>>> from scipy import sparse as sp
>>> C = = sp.csr_matrix(A);
>>> C
<3x3 sparse matrix of type '<type 'NumPy.int32'>'
    with 3 stored elements in Compressed Sparse Row format>
```

如你所见，这个 CSR 矩阵中只存储了三个元素。下面再来看看它实际存储了什么：

```
>>> C.toarray()
array([[1, 0, 0], [0, 2, 0], [0, 0, 3]])
>>> C * C.todense()
matrix([[1, 0, 0], [0, 4, 0], [0, 0, 9]])
```

正如所期待的那样，CSR 矩阵实际上并没有略过所有的 0，用它可以得到与原矩阵一样的计算结果。

```
>>> dot(C, C).todense()
```

8.2.4 优化措施

读者要明白一件事，即每当要在后台构建一个分类器或标注器时，所做的一切都是为

① 译者注：这种方式与 CSR 相对应，因此也叫作**按列压缩存储**。

了执行某种优化历程。下面来对 SciPy 库中所提供的函数做一些基本的了解。从求给定多项式的最小值开始，下面代码所演示的就是 SciPy 库在这方面的一个优化例程。

```
>>> def f(x):
>>>     returnx         return x**2-4
>>> optimize.fmin_bfgs(f,0)
Optimization terminated successfully.
         Current function value: -4.000000
         Iterations: 0
         Function evaluations: 3
         Gradient evaluations: 1
array([0])
```

这里的第一个参数是用来求最小值的函数，而第二个参数则是对该最小值的初始猜测。在这个例子中，我们已经知道了最小值为 0。如果读者还想获取更多详细信息，也可以在这里调用 help()函数：

```
>>> help(optimize.fmin_bfgs)
Help on function fmin_bfgs in module Scipy.optimize.optimize:

fmin_bfgs(f, x0, fprime=None, args=(), gtol=1e-05, norm=inf,
epsilon=1.4901161193847656e-08, maxiter=None, full_output=0, disp=1,
retall=0, callback=None)
    Minimize a function using the BFGS algorithm.

    Parameters
    ---------
    f : callable f(x,*args)
        Objective function to be minimized.
    x0 : ndarray
        Initial guess.
>>> from scipy import optimize
          optimize.fsolve(f, 0.2)
array([ 0.46943096])

>>> def f1 def f1(x,y):
>>>     return x ** 2+  y ** 2 - 4
>>> optimize.fsolve(f1, 0, 0)
array([ 0.])
```

总而言之，读者现在应该对于 SciPy 库中最基本的那些数据结构有了充足的知识准备，并了解了其中的一些最常用的优化技术。目的并不止是要鼓励你去运行一些与机器学习或

自然语言处理相关的应用，而是希望你要超越自己所用的这些 ML 算法所在的数学语境，去看看那些源代码，并试着去理解它们。

具体的实现不仅有助于对算法的理解，而且还能在优化/自定义的过程中去实现自己的需要。

8.3 pandas

下面来讨论一下 pandas 库，这也是 Python 中最令人兴奋的库之一，尤其是对于那些喜欢 R 语言的人来说，想要在 Python 中以更向量化的方式来操作数据，就非它莫属了。我们会在这一节中专门介绍 pandas 库，讨论一些 pandas 框架下的基本数据操作和相关处理。

8.3.1 读取数据

先来看所有数据分析问题中的最重要的任务：如何解析 CSV /其他文件中的数据。

> **提示：**
> 在这里使用的是以下两份文件：
> https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data;
> https://archive.ics.uci.edu/ml/machine-learningdatabases/iris/iris.names。
> 当然，读者也可以自行选择其他 CSV 文件。

首先，要将上述链接中的数据文件下载到本地存储起来，并将其加载到 pandas 数据框架中，具体如下：

```
>>> import pandas as pd
>>> # Please provide the absolute path of the input file
>>> data = pd.read_csv("PATH\\iris.data.txt",header=0")
>>> data.head()
```

	4.9	3.0	1.4	0.2	Iris-setosa
0	4.7	3.2	1.3	0.2	Iris-setosa
1	4.6	3.1	1.5	0.2	Iris-setosa
2	5.0	3.6	1.4	0.2	Iris-setosa

在这里，需要将 CSV 文件读取并存储到 DataFrame 中。在读取 CSV 文件时，有许多选项可用。其中要面对的一个问题就是：所读取的第一行数据在 DataFrame 中会被当作是标题（header)；所以如果要设置真正的标题，就需要将其 header 选项设置为 None，然后再将列表名作为其中的列名来传递给 DataFrame。当然，如果 CSV 文件中已经有了完美形式的标题，那就不需要担心 pandas 库中的标题问题了，因为在默认情况下，它会始终假设第一行数据为标题。之前代码中开头的 0 实际上就被当成了标题行的行号。

所以下面要为之前所用的同一批数据添加一些标题：

```
>>> data = pd.read_csv("PATH\\iris.data.txt", names=["sepal length",
"sepal width", "petal length", "petal width", "Cat"], header=None)
>>> data.head()
```

	sepal length	sepal width	petal length	petal width	Cat
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa

这样一来，就为这个框架创建了一系列临时列名。当然，如果文件本身第一行就是其标题的话，就可以撤下前面的标题选项，这样 pandas 库就会将其检测到的文件首行当作标题。除此之外，作为常见选项的还有 Sep/Delimiter，其主要用于指定列的分隔符。而在读取和清理数据的具体方式上，至少有 20 个不同的优化选项可用，例如删除 Na、删除空白行以及对特定列进行索引等。下面，来看一下其在不同文件类型上的表现：

- read_csv：读取 CSV 文件。
- read_excel：读取 XLS 文件。
- read_hdf：读取 HDFS 文件。
- read_sql：读取 SQL 文件。
- read_json：读取 JSON 文件。

当然，这些都可以在第 2 章中所讨论的那些不同解析方法中找到替代方案。另外，在写文件时也有相同数量的选项是可用的。

接下来看看 pandas 框架具体有多大的能力。如果你是个 R 程序员，想必会喜欢看到类似 R 语言中的摘要和标题选项。

```
>>> data.describe()
```

这里的 describe()函数将会为你提供一份关于每一列以及相关唯一值的简要信息。

```
>>> sepal_len_cnt=data['sepal length'].value_counts()
>>> sepal_len_cnt

5.0     10
6.3      9
6.7      8
5.7      8
5.1      8
dtype: int64
>>>data['Iris-setosa'].value_counts()
Iris-versicolor    50
Iris-virginica     50
Iris-setosa        48
dtype: int64
```

为了满足 R 语言的爱好者，现在处理向量也可以通过这样的东西来查看列中的每个值：

```
>>>data['Iris-setosa'] == 'Iris-setosa'
0      True
1      True

147    False
148    False
Name: Iris-setosa, Length: 149, dtype: bool
```

现在还可以对 DataFrame 进行过滤。例如这里的 setosa 只能匹配与 Iris-setosa 相关的条目。

```
>>> sntsosa=data[data['Cat'] == 'Iris-setosa']
>>> sntsosa[:5]
```

这其实就是典型的 SQL 分组函数。其他聚合函数的情况也差不多是这样。

> **提示：**
> 你可以通过下面链接来查看道琼斯的数据：
> https://archive.ics.uci.edu/ml/machine- learningdatabases/00312/。

8.3.2 数列

pandas 库中也有一种按日期索引的简洁方式，并支持日后按照各种时间顺序对其进行

数据分析。这样做最大的优点就是只要按日期对数据进行了索引，那些最令人痛苦的日期操作就是一个命令的事。下面，就来具体看看序列数据的处理，例如这里有几支股票的股价数据及其每周开盘价和收盘价的变化。①

```
>>> import pandas as pd
>>> stockdata = pd.read_csv("dow_jones_index.data",parse_dates=['date'],
index_col=['date'],nrows=100)
>>> stockdata.head()
```

date	quarter	stock	open	high	low	close	volume	percent_ change_ price
01/07/2011	1	AA	$15.82	$16.72	$15.78	$16.42	239655616	3.79267
01/14/2011	1	AA	$16.71	$16.71	$15.64	$15.97	242963398	−4.42849
01/21/2011	1	AA	$16.19	$16.38	$15.60	$15.79	138428495	−2.47066

```
>>> max(stockdata['volume'])
    1453438639
>>> max(stockdata['percent_change_price'])
    7.62173999999999991
>>> stockdata.index
<class 'pandas.tseries.index.DatetimeIndex'>
[2011-01-07, ..., 2011-01-28]
Length: 100, Freq: None, Timezone: None
>>> stockdata.index.day
array([ 7, 14, 21, 28, 4, 11, 18, 25, 4, 11, 18, 25, 7, 14, 21, 28, 4,11,
18, 25, 4, 11, 18, 25, 7, 14, 21, 28, 4])
```

上述命令给出的是每个日期的日信息。②

```
>>> stockdata.index.month
```

上述命令将会按月列出不同的值。

```
>>> stockdata.index.year
```

上述命令将会按年列出不同的值。

也可以通过一个名为 resample 的函数按照自己的想法对数据进行聚合。它的选项包括

① 译者注：在这里，为了让表格与代码保持一致，就不对表头部分进行翻译了，下同。

② 译者注：此处原文似乎与代码对不上，如果按照原文 day of the week for each date，上面调用的应该是 stockdata.index.dayofweek。

sum、mean、median、min 和 max。

```
>>> import numpy as np
>>> stockdata.resample('M', how=np.sum)
```

8.3.3 列转换

假设现在想要过滤某些列或添加列，就可以用刚才提供的列之列表来充当 axis1 参数来实现它。可以像下面这样来删除数据框架中的某列数据：

```
>>> stockdata.drop(["percent_change_volume_over_last_wk"],axis=1)
```

下面来过滤掉一些自己不想要的列，并且将工作限制在某一组列中。为此，可以像这样新建一个 DataFrame：

```
>>> stockdata_new = pd.DataFrame(stockdata, columns=["stock","open","high"
,"low","close","volume"])
>>> stockdata_new.head()
```

还可以执行一些类似于 R 语言中的列操作，如说重命名列。另外，也可以做这样的事：

```
>>> stockdata["previous_weeks_volume"] = 0
```

这会将目标列中的所有值都更改为 0，可以有条件地执行这样的操作，并在其中创建一些派生变量。

8.3.4 噪声数据

通常情况下，数据科学家每天的日常生活都是从数据清洗开始。这项任务包括移除噪声、清除一些不想要的文件、确保日期格式的正确、忽略干扰性记录并处理丢失的值。总之，数据清理所占用的时间段往往是最大的，其他活动并非如此。

在实际的应用场景中，数据在大多数情况下都是混乱的，我们必须处理其中的缺失值、空值、Na 以及其他格式问题。因此，任何处理数据的程序库的一个主要功能就是要能处理上述问题并能以有效的方式解决它们。对于这其中的一些问题，pandas 库倒也提供了一些令人惊叹的功能。

```
>>> stockdata.head()
>>> stockdata.dropna().head(2)
```

通过上面的命令，清除掉了目标数据中所有的 Na。

date	quarter	stock	open	high	low	close	volume	percent_ change_ price
01/14/2011	1	AA	$16.71	$16.71	$15.64	$15.97	242963398	−4.42849
01/21/2011	1	AA	$16.19	$16.38	$15.60	$15.79	138428495	−2.47066
01/28/2011	1	AA	$15.87	$16.63	$15.82	$16.13	151379173	1.63831

另外，你应该也注意到一些值之前有个$符，它会给数字操作带来一些困难。下面要来解决这一麻烦，因为不这样做，它会干扰结果（例如可能会导致$ 43.86 不是最值）。

```
>>> import numpy
>>> stockdata_new.open.describe()
count           100
unique           99
top          $43.86
freq              2
Name: open, dtype: object
```

还可以针对两列数据执行一些操作，并从中生成一个新变量：

```
>>> stockdata_new.open = stockdata_new.open.str.replace('$', '').convert_
objects(convert_numeric=True)
>>> stockdata_new.close = stockdata_new.close.str.replace('$', '').
convert_objects(convert_numeric=True)
>>> (stockdata_new.close - stockdata_new.open).convert_objects(convert_
numeric=True)
>>> stockdata_new.open.describe()
count      100.000000
mean        51.286800
std         32.154889
min         13.710000
25%         17.705000
50%         46.040000
75%         72.527500
max        106.900000
Name: open, dtype: float64
```

也可以执行一些算术运算，并为此创建新的变量。

```
>>> stockdata_new['newopen'] = stockdata_new.open.apply(lambda x: 0.8 * x)
>>> stockdata_new.newopen.head(5)
```

另外，还可以用下面这种方式来过滤某列值中的数据。例如，从上面的股票值中过滤出其中一个公司的数据集。

```
>>> stockAA = stockdata_new.query('stock=="AA"')
>>> stockAA.head()
```

总而言之，这一节介绍了一系列 pandas 库中与数据读取、清理、操纵以及聚合有关的实用函数。下一节来看看如何利用这些数据框架产生针对这些数据的可视化图表。

8.4 matplotlib

matplotlib 是 Python 语言环境中一个非常受欢迎的可视化库。接下来，将为你介绍一些最常用的可视化应用。首先导入这个库：

```
>>> import matplotlib
>>> import matplotlib.pyplot as plt
>>> import numpy
```

下面，要利用运行在道琼斯指数上的数据集来执行一些可视化操作。之前，已经拥有了公司“AA”的股票数据。下面要为一家新公司 CSCO 再创建一个数据框架，然后围绕着它进行一些绘制工作：

```
>>> stockCSCO = stockdata_new.query('stock=="CSCO"')
>>> stockCSCO.head()
>>> from matplotlib import figure
>>> plt.figure()
>>> plt.scatter(stockdata_new.index.date,stockdata_new.volume)
>>> plt.xlabel('day') # added the name of the x axis
>>> plt.ylabel('stock close value') # add label to y-axis
>>> plt.title('title') # add the title to your graph
>>> plt.savefig("matplot1.jpg")  # savefig in local
```

还可以将上面绘制的图保存为 JPEG / PNG 文件，只需要调用 savefig()函数，例如：

```
>>> plt.savefig("matplot1.jpg")
```

8.4.1 子图绘制

子图绘制法是布局整体绘制的最佳方式。这就像在一块画布上工作，可以在上面不止有一轮绘制，而是多轮绘制的叠加。下例将会进行四轮绘制。通过参数 numrow、numcol 来定义画布，并在下一个参数指定绘制的编号。

```
>>> plt.subplot(2, 2, 1)
>>> plt.plot(stockAA.index.weekofyear, stockAA.open, 'r--')
```

```
>>> plt.subplot(2, 2, 2)
>>> plt.plot(stockCSCO.index.weekofyear, stockCSCO.open, 'g-*')
>>> plt.subplot(2, 2, 3)
>>> plt.plot(stockAA.index.weekofyear, stockAA.open, 'g--')
>>> plt.subplot(2, 2, 4)
>>> plt.plot(stockCSCO.index.weekofyear, stockCSCO.open, 'r-*')
>>> plt.subplot(2, 2, 3)
>>> plt.plot(x, y, 'g--')
>>> plt.subplot(2, 2, 4)
>>> plt.plot(x, y, 'r-*')
>>> fig.savefig("matplot2.png")
```

以上代码的输出如图 8-1 所示。

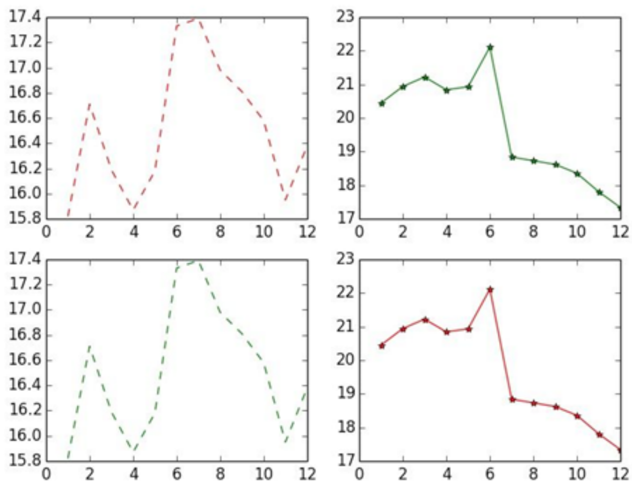


图 8-1

也可以用一些更优雅的做法来进行多次绘制！

```
>>> fig, axes = plt.subplots(nrows=1, ncols=2)
>>> for ax in axes:
>>>      ax.plot(x, y, 'r')
>>>      ax.set_xlabel('x')
>>>      ax.set_ylabel('y')
>>>      ax.set_title('title');
```

如你所见，可以有很多种编写代码的方式，这更像是在通用的 Python 环境中，按自身所想的方式处理不同方面的绘制问题。

8.4.2　添加坐标轴

可以通过调用 addaxis()函数来为图片添加轴。通过向图片中添加轴，可以定义一块属

于自己的绘制区域。addaxis()的调用参数如下：

```
*rect* [*left*, *bottom*, *width*, *height*]
>>> fig = plt.figure()
>>> axes = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # left, bottom, width, height
(range 0 to 1)
>>> axes.plot(x, y, 'r')
```

下面，来绘制一些最常用的图形类型。这里最棒的一件事是，大部分的参数，像标题和标签这样的，和之前的工作方式依然是一样的。只是绘制的种类会有所变化。

如果希望通过坐标轴来添加一个 x 标签，一个 y 标签和一个标题，其命令如下：

```
>>> fig = plt.figure()
>>> ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])
>>> ax.plot(stockAA.index.weekofyear,stockAA.open,label="AA")
>>> ax.plot(stockAA.index.weekofyear,stockCSCO.open,label="CSCO")
>>> ax.set_xlabel('weekofyear')
>>> ax.set_ylabel('stock value')
>>> ax.set_title('Weekly change in stock price')
>>> ax.legend(loc=2); # upper left corner
>>> plt.savefig("matplot3.jpg")
```

请读者自行尝试编写上述代码，并观察其输出！如图 8-2 所示。



图 8-2

8.4.3 散点图绘制

绘制图最简单的形式之一就是针对 x 轴的不同值来绘制 y 轴的点。下面示例会尝试在一幅散点图中捕捉股价每周的变化：

```
>>> import matplotlib.pyplot as plt
>>> plt.scatter(stockAA.index.weekofyear,stockAA.open)
>>> plt.savefig("matplot4.jpg")
>>> plt.close()
```

8.4.4 条形图绘制

为直观起见，可以用下面这种条形图来反映 y 轴上的值相对于 x 轴的分布情况。下面就具体来示范一下如何通过绘制条形图来显示数据。

```
>>> n = 12
>>> X = np.arange(n)
>>> Y1 = np.random.uniform(0.5, 1.0, n)
>>> Y2 = np.random.uniform(0.5, 1.0, n)
>>> plt.bar(X, +Y1, facecolor='#9999ff', edgecolor='white')
>>> plt.bar(X, -Y2, facecolor='#ff9999', edgecolor='white')
```

8.4.5 3D 绘图

也可以用 matplotlib 库来构建一些壮观的三维可视化图表。下面就来示范一下如何用 matplotlib 库来创建三维绘制图。

```
>>> from mpl_toolkits.mplot3d import Axes3D
>>> fig = plt.figure()
>>> ax = Axes3D(fig)
>>> X = np.arange(-4, 4, 0.25)
>>> Y = np.arange(-4, 4, 0.25)
>>> X, Y = np.meshgrid(X, Y)
>>> R = np.sqrt(X**2+ + Y**2)
>>> Z = np.sin(R)
>>> ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='hot')
```

8.5 参考资料

我们希望鼓励读者去阅读一下下面链接中的内容，以便更详细地了解这些库的方方面面，进而获得更多的资源。

- http://www.NumPy.org/。
- http://www.Scipy.org/。
- http://pandas.pydata.org/。

- http://matplotlib.org/。

8.6 小结

本章对一些最基本的 Python 库进行了简单的概要性介绍，这些库在处理文本及其他数据时能完成很多重要工作。NumPy 库可以帮助用户处理数值计算的问题，并提供解决这类问题所需的一些数据结构。SciPy 库则主要用于处理科学计算，业界有许多各种各样的 Python 库都会用到它。我们学习了这些函数和数据结构的具体用法。

另外，本章还对 pandas 库进行了介绍，这是一个非常有效率的针对数据操纵的程序库，并在最近这段时间内发挥出了巨大的优势。最后，还带你快速浏览了 Python 环境中最常用的可视化库：matplotlib。

下一章将会把焦点转向社交媒体。还会介绍如何从一些常见的社交网络捕获相关的数据，并借此来产生针对社交媒体的有意义的见解。

第 9 章
Python 中的社交媒体挖掘

这一章来讨论一下社交媒体。虽然这方面的内容与 NLTK/NLP 没有直接关系，但社交数据也是一种非常丰富的非结构化文本的数据源。作为 NLP 爱好者，我们应该掌握一些处理社交数据的技能。本章将会探讨如何从一些目前最受欢迎的社交媒体平台中收集到相关数据。还会介绍如何利用 Python API 来从 Twitter、Facebook 等社交媒体中收集数据。还会探讨一些在社交媒体挖掘领域中最常见的用例，例如热门话题、情绪分析等。

我们在前面的章节中已经学习了许多与自然语言处理和机器学习相关的概念性话题。本章将会试着围绕一些社交数据来构建一些应用程序。本章还提供了一些针对社交数据处理的最佳实践，并以可视化图形的方式来查看这些社交数据。

社交媒体都会存在一个基础性的图结构，而大多数基于图结构的问题都可以被表述成某种信息流问题，并找出该图结构中最繁忙的节点。像热门话题、影响力检测以及情绪分析这些问题都是很好的例子。下面就通过这些具体的用例，围绕社交网络来构建一些酷炫的应用程序吧。

在阅读完本章之后，我们希望你能掌握以下内容。

- 知道如何用相关 API 收集任意社交媒体中的数据。
- 学会如何用某种结构化格式来表述数据，并以此构建出一些很棒的应用程序。
- 可以为社交数据绘制可视化图形，并能对其进行有意义的观察。

9.1 数据收集

本章最重要的目标是要介绍如何在一些业界最常见的社交网络之间进行数据收集。

本章主要以 Twitter 和 Facebook 为实验对象，为你详细、充分地介绍与这两个社交媒体有关的 API 信息，以及如何有效地利用它们来获取相关数据。此外，还将讲解与废弃数据相关的数据字典，以及如何利用目前所学到的知识来构建一些酷炫的应用程序。

Twitter

先从目前最流行、最开放的且完全公开的社交媒体开始入手。这实际上就意味着可能要去收集整个 Twitter 流中的信息，但这是要付费的，但可以免费捕获其中百分之一的信息。在商业背景下，对于那些想要了解公众情绪、新兴话题这类信息的人来说，Twitter 是一个非常丰富的信息资源。

下面，就来解决如何从 tweet 中获取与用例相关信息这一主要问题吧。

> **提示：**
> 下面链接中列出了许多 Twitter 程序库的代码仓库[1]。当然，这些程序库都没有经过 Twitter 官方的验证，但它们都可以基于 Twitter API 来运行:
> https://dev.twitter.com/overview/api/twitterlibraries。

具有这方面功能的 Python 库绝对超过 10 个，所以可以选择任意一个自己自己喜欢的。由于我自己通常会选择 Tweepy，所以这里将会使用它来完成本书中的示例。这些程序库大部分都是一些 Twitter API 的封装器，因此它们的参数和签名也大致相同。

安装 Tweepy 最简单的方法是使用 pip：

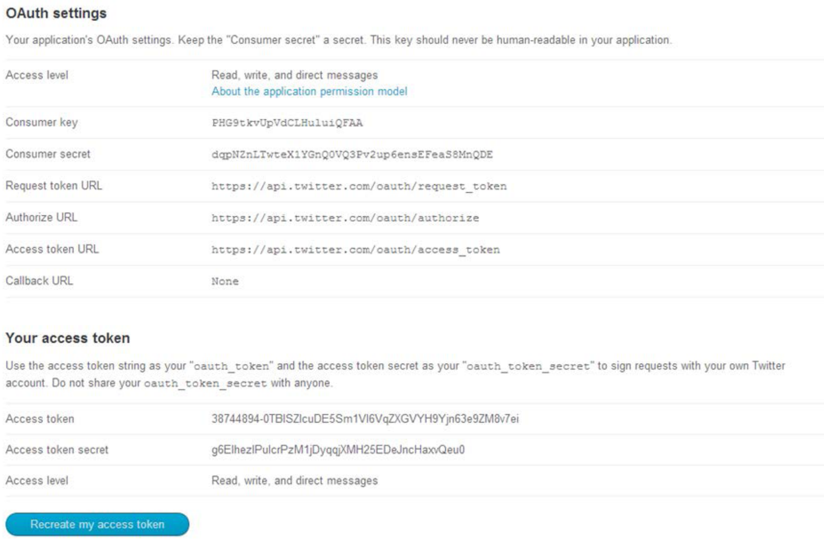
```
$ pip install tweepy
```

> **提示：**
> 源代码安装是比较难的一种安装方式，Tweepy 在 github 上的链接如下:
> https://github.com/tweepy/tweepy。

当然，如果想要让 Tweepy 能够正常工作，必须在 Twitter 上创建一个开发者帐户，为将要创建的应用获取访问令牌。在完成这些动作之后，就会得到属于自己的的资格验证以及这些验证信息下面的密钥。也可以在 https://apps.twitter.com/app/new 中注册并获取令牌。

① 译者注：经译者查证，该链接已经失效，读者可以通过访问 https://dev.twitter.com/resources/twitter-libraries 来获取相关内容。

下图显示的就是一个访问令牌的快照：



先来看一个非常简单的例子：通过 Twitter 信息流的 API 来收集数据。使用 Tweepy 来捕获 Twitter 流，收集其中所有与给定关键字相关的 tweet：

```
tweetdump.py
>>> from tweepy.streaming import StreamListener
>>> from tweepy import OAuthHandler
>>> from tweepy import Stream
>>> import sys
>>> consumer_key = 'ABCD012XXXXXXXXx'
>>> consumer_secret = 'xyz123xxxxxxxxxxxxx'
>>> access_token = '000000-ABCDXXXXXXXXXXX'
>>> access_token_secret ='XXXXXXXXXgaw2KYz0VcqCO0F3U4'
>>> class StdOutListener(StreamListener):
>>>     def on_data(self, data):
>>>         with open(sys.argv[1],'a') as tf:
>>>             tf.write(data)
>>>         return
>>>     def on_error(self, status):
>>>         print(status)
>>> if __name__ == '__main__':
>>>     l = StdOutListener()
>>>     auth = OAuthHandler(consumer_key, consumer_secret)
>>>     auth.set_access_token(access_token, access_token_secret)
>>>     stream = Stream(auth, l)
>>>     stream.filter(track=['Apple watch'])
```

上述代码使用了与 Tweepy 示例相同的代码，并作了稍许修改。这个例子示范了如何使用 Twitter 信息流的 API，跟踪的关键词是 **Apple Watch**。Twitter 信息流的 API 在这里实际提供的就是在 Twitter 信息流中执行搜索的功能，可以用该 API 查看其信息流中最多百分之一的信息。

对于上述代码，主要需要理解的部分是头 4 行和最后 4 行。在初始化的那几行代码中，指定的是上一节中生成的访问令牌和其他相关密钥。而在最后 4 行中，创建了一个针对信息流的监听器。特别在最后一行，使用了 stream.filter 来过滤 twitter，以便设置要跟踪的关键字。这里可以一次设置多个关键字。在这个例子中，运行结果中包含所有与 Apple Watch 这个词相关的 tweet。

接下来的这个示例，要将上面收集的 tweet 载入，带你来看看 tweet 的结构，并探讨如何从中提取出有意义的信息。通常情况下，tweet JSON 在结构上应该是这样的：

```
{
"created_at":"Wed May 13 04:51:24 +0000 2015",
"id":598349803924369408,
"id_str":"598349803924369408",
"text":"Google launches its first Apple Watch app with News & Weather
http:\/\/t.co\/o1XMBmhnH2",
"source":"\u003ca href=\"http:\/\/ifttt.com\" rel=\"nofollow\"\
u003eIFTTT\u003c\/a\u003e",
"truncated":false,
"in_reply_to_status_id":null,
"user":{
"id":1461337266,
"id_str":"1461337266",
"name":"vestihitech \u0430\u0432\u0442\u043e\u043c\u0430\u0442",
"screen_name":"vestihitecha",
"location":"",
"followers_count":20,
"friends_count":1,
"listed_count":4,
""statuses_count":7442,
"created_at":"Mon May 27 05:51:27 +0000 2013",
"utc_offset":14400,
},
,
"geo":{ "latitude" : 51.4514285, "longitude"=-0.99
}
"place":"Reading, UK",
"contributors":null,
```

```
"retweet_count":0,
"favorite_count":0,
"entities":{
"hashtags":["apple watch", "google"
],
"trends":[
],
"urls":[
{
"url":"http:\/\/t.co\/o1XMBmhnH2",
"expanded_url":"http:\/\/ift.tt\/1HfqhCe",
"display_url":"ift.tt\/1HfqhCe",
"indices":[
66,
88
]
}
],
"user_mentions":[
],
"symbols":[
]
},
"favorited":false,
"retweeted":false,
"possibly_sensitive":false,
"filter_level":"low",
"lang":"en",
"timestamp_ms":"1431492684714"
}
]
```


9.2 数据提取

在数据提取的过程中，有一些最常用到的字段。

- text：即用户所提供的 tweet 内容。
- user：即用户的一些主要属性，如用户名、所在地和照片等。
- Place：即发布 tweet 的地方，也是地理坐标。
- Entities：即用户在其 tweet 中所附加的有效主题标签和主题。

在实践中，上述每个属性都可以成为执行社交媒体挖掘的实验用例。下面，来看看应该如何获取这些属性，并将其转换成某种可读性更好的形式，或者对其进行其他相关的处理：

```
Source: tweetinfo.py
>>> import json
>>> import sys
>>> tweet = json.loads(open(sys.argv[1]).read())

>>> tweet_texts = [ tweet['text']\
                                for tweet in tweet ]
>> >tweet_source = [tweet ['source'] for tweet in tweet]
>>> tweet_geo = [tweet['geo'] for tweet in tweet]
>>> tweet_locations = [tweet['place'] for tweet in tweet]
>>> hashtags = [ hashtag['text'] for tweet in tweet for hashtag in
tweet['entities']['hashtags'] ]
>>> print tweet_texts
>>> print tweet_locations
>>> print tweet_geo
>>> print hashtags
```

正如所期待的那样，上述代码会输出 4 个列表，其中，tweet_texts 中包含的是所有的 tweet 内容，此外还有该 tweet 所在的位置和主题标签。

提示：
这段代码只是用 json.loads()加载了一段以 JSON 格式生成的输出。建议读者使用像 Json Parser（http:// json.parser.online.fr/）这样的线上工具来帮助你理解这段 JSON 输出的含义，以及它的属性（键和值）内容。

如果仔细观察的话，会发现这段 JSON 被分成了不同的层次，其中有些属性，如 text 就只有一个直接值，而有些属性有更多的嵌套信息。这就是为什么如果想要查看 hashtag 属性，就必须迭代上几个层次，而对于 test 属性，就只需直接获取值即可。由于文件实际是一个 tweet 列表，所以必须要通过迭代该列表才能获取到所有的 tweet，而这里每个 tweet 对象的结构和上面那个 tweet 示例是基本相同的。

热门话题

如果想要在上面这种设置条件下查找出当前热门的话题。最简单的一种方法就是查看所有 tweet 中单词的频率分布情况。因为目前已经有了一个包含所有 tweet 的 tweet_text 列表：

```
>>> import nltk
>>> from nltk import word_tokenize,sent_tokenize
>>> from nltk import FreqDist
>>> tweet_tokens = []

>>> for tweet in tweet_text:
>>>     tweet_tokens.append(word_tokenize(tweet))
>>> Topic_distribution = nltk.FreqDist(tweet_tokens)
>>> Freq_dist_nltk.plot(50, cumulative=False)
```

除此之外，还有一个更复杂一些的方法，就是利用第 3 章“词性标注”中所学到的词性标注器。其原理是：主题在大多数情况下都是一些名词或实体项，所以，可以对其执行相同的处理。上面的代码已经读取了每个 tweet 并对其进行了标识化，下面以 POS 为过滤器，只选取其中的名词来充当主题：

```
>>> import nltk
>>> Topics = []
>>> for tweet in tweet_text:
>>>     tagged = nltk.pos_tag(word_tokenize(tweet))
>>>     Topics_token = [word for word,pos in ] in tagged if pos in ['NN','NNP']
>>>     print Topics_token
```

如果想让这个例子再酷炫一点，还可以跨时段地收集一些 tweet，然后为其生成绘制图。这样就会得到一份含义非常清楚的热门报告。例如，之前正在寻找的是与“Apple Watch”相关的数据。这个词应该会在苹果推出 Apple Watch 的那一天，以及该产品开始销售的那一天达到高峰。但更感兴趣的是那几天除了这些主题，还出现了什么样的主题，这些主题随着时间呈现出了何种讨论趋势。

9.3 地理可视化

地理位置的可视化也是社交媒体的常见应用之一。在 tweet 结构中，会看到 geo、longitude 和 latitude 这三个与之相关的属性。通过访问这些属性，就能利用像 D3 这样的可视化程序库做出图 9-1 所示的内容。

在这个例子中看到的只是其中一种可视化实现，它呈现的是全美 tweet 的可视化分布。从中可以清楚地看出纽约这些东部地区的使用强度正在增长。现在，公司可以通过对客户进行类似的分析来厘清其客户群喜欢待在哪些地方。除此之外，还能根据情绪来对 tweet 进行文本挖掘，以此来推断哪几个州的客户对公司不太满意等情况。

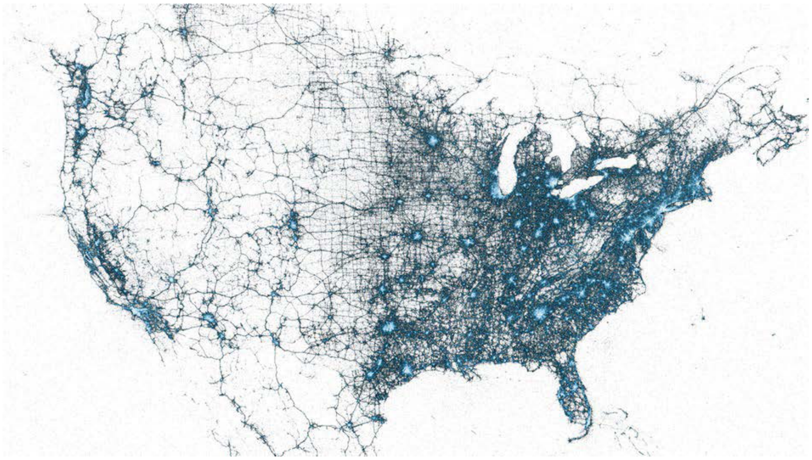


图 9-1

9.3.1 影响力检测

在社交图语境中，对具有重要影响力的社交图中的重要节点进行检测也是一个很重要的问题。因此，如果我们手里有数百万条关于我们公司的 tweet，那么其中一个重要用例就是要收集该社交媒体中最有影响力的那些客户，然后针对他们进行品牌推广、营销或改善他们的参与度。

如果具体到 Twitter 的情况，就得回到图论和 PageRank 的概念上来，其中对于一个给定的节点，如果其出度（outdegree）比例比入度（indegree）高，该节点就是具有影响力的。原因非常直观，拥有较多关注者的人显然通常要比他们所关注的人有影响力。有一家名为 KLOUT（https://klout.com/）的公司一直很专注于这类问题。所以下面就来写一个最基本而且直观的算法来为这些公司评个分：

```
>>> klout_scores = [ (tweet['user']['followers_count]/tweet['user']
['friends_count'],tweet['user']) for tweet in tweet ]
```

当然在这些以 Twitter 为基础的例子中，我们所修改的字段内容始终是完全相同的。其实也可以用 Facebook 帖子来示范如何构建热门话题。同样地，也可以示范如何对 Facebook 用户、所发帖的地理位置及其影响力进行可视化处理。事实上，下一节就要来看看这些应用在 Facebook 中会有哪些变化。

9.3.2 Facebook

Facebook 的私人化程度要更高一些，有些接近于私人式的社交网络。Facebook 出于隐

私和安全方面的考虑，通常不会允许我们收集用户的订阅内容/帖子。因此，Facebook 的图结构 API 只能以有限的方式来反馈给定页面，关于这方面的内容，推荐读者去阅读一下 https://developers.facebook.com/docs/graph-api/using-graph-api/v2.3 中的说明，以加深理解。

接下来要解决的一个问题是如何用 Python 来访问这些图结构 API 并加以应用。业界围绕着 Facebook API 编写了很多封装器，这里使用的是最常见的 Facebook SDK：

```
$ pip install facebook-sdk
```

> **小技巧：**
> 也可以到以下链接中获取相应的安装包:
> https://github.com/Pythonforfacebook/facebook-sdk。

然后，下一步就是要获取应用程序的访问令牌了。Facebook 会将每个 API 调用都视为一个应用程序。因此即使是数据收集这一步骤，也需要将其伪装成一个应用程序。

> **提示：**
> 如果想要获得属于自己的访问令牌，请访问:
> https://developers.facebook.com/tools/explorer。

现在所有设置都完成了！下面先来试试一个使用度最高的 Facebook 的图结构 API。在该 API 中，Facebook 提供的是一个针对页面、用户、事件、地点等信息的基于图结构的搜索功能。这种情况下，获得相关帖子的过程变成了两个阶段，必须先找到与自己感兴趣的主题相关的 pageid / userid，然后才能访问到该页面的订阅内容。下面就要来实现一个简单的用例，就是在某公司的官方页面上寻找客户投诉。其具体做法如下：

```
>>> import facebook
>>> import json

>>> fo = open("fdump.txt",'w')
>>> ACCESS_TOKEN = 'XXXXXXXXXXX' # https://developers.facebook.com/tools/explorer
>>> fb = facebook.GraphAPI(ACCESS_TOKEN)
>>> company_page = "326249424068240"
>>> content = fb.get_object(company_page)
>>> fo.write(json.dumps(content))
```

这段代码要将访问令牌附加到 Facebook 的图结构 API 中，这样才能通过该 API 对 Facebook 做一个 REST 调用。但前提是必须要先获得给定页面的 ID。下面来看看要附加的访问令牌，其代码如下：

```
"website":"www.dimennachildrenshistorymuseum.org",
"can_post":true,
```

```
"category_list":[
{
"id":"244600818962350",
"name":"History Museum"
},
{
"id":"187751327923426",
"name":"Educational Organization"
}
],
"likes":1793,
},
"id":"326249424068240",
"category":"Museum/art gallery",
"has_added_app":false,
"talking_about_count":8,
"location":{
"city":"New York",
"zip":"10024",
"country":"United States",
"longitude":-73.974413,
"state":"NY",
"street":"170 Central Park W",
"latitude":40.779236
},
"is_community_page":false,
"username":"nyhistorykids",
"description":"The first-ever museum bringing American history to life
through the eyes of children, where kids plus history equals serious
fun! Kids of all ages can practice their History Detective skills at
the DiMenna Children's History Museum and:\n\n\u2022 discover the past
through six historic figure pavilions\n\n\u2022!",
"hours":{
""thu_1_close":"18:00"
},
"phone":"(212) 873-3400",
"link":"https://www.facebook.com/nyhistorykids",
"price_range":"$ (0-10)",
"checkins":1011,
"about":"The DiMenna Children' History Museum is the first-ever museum
bringing American history to life through the eyes of children. Visit it
inside the New-York Historical Society!",
"name":"New-York Historical Society DiMenna Children's History Museum",
"cover":{
"source":"https://scontent.xx.fbcdn.net/hphotos-xpf1/t31.0-8/s720x720/104
```

```
9166_672951706064675_339973295_o.jpg",
"cover_id":"672951706064675",
"offset_x":0,
"offset_y":54,
"id":"672951706064675"
},
"were_here_count":1011,
"is_published":true
},
```


如你所见，这里所展现的 Facebook 数据模式与 Twitter 非常类似。接下来，可以来看看这个用例究竟请求到了怎样的信息。在大多数情况下，用户的 post、category、name、about 和 likes 都是它的重要字段。当然，这个例子所演示的是一个博物馆的页面，而在更商业化的用例中，公司页面上应该会有一个长长的帖子列表和其他有用的信息，它们都可以提供很好的观察点。

假设组织 xyz.org 在 Facebook 上有一个专属页面，而现在想要了解有哪些用户在页面上投诉了我们。这对于投诉分类这样的应用来说也是一个很好的用例。目前实现这一应用的方式非常简单。需要先在 fdump.txt 中查找一组关键字，然后其操作的复杂程度就与第 6 章使用文本分类算法来进行评分的方法差不多了。

现在来看另一个用例：查找感兴趣的主题，并在其结果页面中查看开放性帖子及其评论。这个功能与直接用 Facebook 主页上的图形搜索栏来搜索非常类似。但用编程的方式来做这件事的好处是可以自行主导这些搜索，并可以对每个页面上的用户评论进行递归搜索。下面来看看搜索用户数据的代码：

User search

```
>>> fb.request("search", {'q' : 'nitin', 'type' : 'user'})
Place based on the nearest location.
>>> fb.request("search", {'q' : 'starbucks', 'type' : 'place'})
Look for open pages.
>>> fb.request("search", {'q' : 'Stanford university', 'type' : page})
Look for event matching to the key word.
>>> fb.request("search", {'q' : 'beach party', 'type' : 'event'})
```

一旦将所有的相关数据都转换成了结构化格式，接下来就可以运用在探讨 NLP 和机器学习这些主题时所学到的那些概念了。下面选择用同一个用例来查找帖子，不过这回主要找到的是该 Facebook 页面上的投诉信息。

假设现在得到了以下格式的数据[①]：

① 译者注：考虑到原文讨论的是英文环境下的文本处理，其处理逻辑跟中文大不相同，所以数据文本还是不翻译为好。

Userid	FB Post
XXXX0001	The product was pathetic and I tried reaching out to your customer care, but nobody responded
XXXX002	Great work guys
XXXX003	Where can I call to get my account activated ??? Really bad service

在这里，我们等于是回到了第 6 章所用过的那个示例中，当时构建了一个文本分类器，并用它来检测某段 SMS（消息文本）是否为垃圾邮件。类似地，在这里也可以利用这些帖子中的数据来创建训练数据，在这里需要这些手动标注器标注出那些带投诉内容的评论，其他则一概忽略。一旦得到了这些重要的训练数据，就可以构建出相同的文本分类器了：

```
fb_classification.py
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> vectorizer = TfidfVectorizer(min_df=2, ngram_range=(1, 2), stop_
words='english',  strip_accents='unicode',  norm='l2')
>>> X_train = vectorizer.fit_transform(x_train)
>>> X_test = vectorizer.transform(x_test)

>>> from sklearn.linear_model import SGDClassifier
>>> clf = SGDClassifier(alpha=.0001, n_iter=50).fit(X_train, y_train)
>>> y_pred = clf.predict(X_test)
```

接下来，假设这里只有三种样本。将其中的第一种和第三种标注为投诉类信息，第二种则是非投诉类信息。尽管是以相同的方式构建了一个单元或二元模型的向量化器，但事实上这里是可以用相同的处理过程来构建一个分类器的，因为忽略了这里的一些预处理步骤。你可以在这里执行与第 6 章中相同的处理过程。当然在某些情况下，想要得到像上面这样的训练数据是非常困难或者代价高昂的。所以对于其中的一些情况，也可以采用像文本聚类或主题建模这样的无监督型算法。除此之外，另一种做法是采用一些开放使用的不同数据集，然后用它来构建模型并应用。例如在同一个用例中，也可以直接以 Web 的方式来抓取一些可用的客户投诉，并将其用作模型中的训练数据。这也是一种很好的获取相关标签数据的替代做法。

9.3.3 有影响力的朋友

找出你社交关系图中最有影响力的人也是社交媒体中一个常见的用例。对于我们来说这个问题就是要找出一个在图结构中拥有大量入向链接和外向链接的节点，它就被认为是有影响力的节点。

在商业背景下，相同的问题可能就是要找出最有影响力的客户，并以他们为目标来推销产品。

下面就来看一下寻找有影响力的朋友的代码：

```
>>> friends = fb.get_connections("me", "friends")["data"]
>>> print friends
>>> for frd in friends:
>>>     print fb.get_connections(frd["id"],"friends")
```

一旦手里有了一个包含自身所有朋友以及这些朋友的共同朋友的列表，就可以创建一个这样的数据结构：

源节点	目标节点	存在链接数
朋友 1	朋友 2	1
朋友 1	朋友 3	1
朋友 2	朋友 3	0
朋友 1	朋友 4	1

这是一种可用来生成某种网络关系的数据结构，它非常有利于对社交关系图进行可视化。这里使用的 D3 库，但 Python 社区中还有一个名为 NetworkX（https://networkx.github.io/）的库，也可以用来生成可视化的图结构，如图 9-2 所示。要想生成可视化的图结构，首先要基于之前的信息弄清楚谁是谁的朋友，以此来创建一个邻接矩阵。

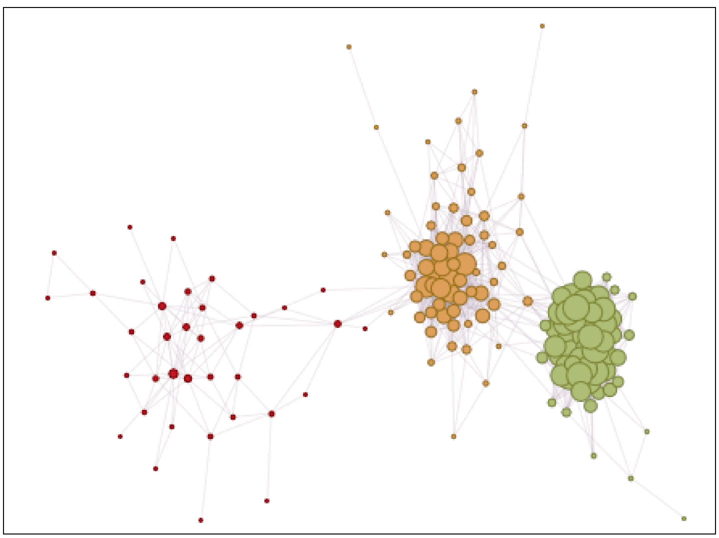


图 9-2

9.4 小结

本章带你接触到了一些当下最流行的社交网络，并学习了如何使用 Python 来获取它们的数据。在这个过程中，我们带你了解了它们的数据结构和各类属性数据，并探索了其 API 所提供的不同选项。

另外，也探讨了一些在社交媒体挖掘领域中最常见的用例。这些用例包括热门话题提取、影响者检测，信息流分析等。还对其中的一些用例进行了可视化处理。除此之外，还运用了上一章所学到的知识，用 NLTK 来完成一些主题获取和实体提取的任务，并用 scikit-learn 对一些投诉信息进行了分类。

最后，建议读者可以再寻找一些其他的社交网络环境来执行相同的用例，并自行探索它们。这些社交网络大部分都会提供一组数据 API，而且其中绝大多数是开放的，足以做一些有趣的分析。如果你想实现本章所学习到的这些应用，就需要了解如何用这些 API 来获取数据，然后如何运用之前章节中所学到的一些概念。期待读者在学习这一切的过程中能想出更多的用例，和一些有趣的社交媒体分析。

第 10 章
大规模文本挖掘

本章打算再回顾之前章节中提到的一些程序库，但这回要谈的是如何在大数据环境中大规模地使用这些库。因此，本章会假设读者对于 Hadoop+Hive 这样的大数据框架已经有了一定的了解。在此基础之上，我们会对一些 Python 库进行一些相应的探讨，例如 NLTK、scikit-learn 和 pandas 这几个库都可以被应用于带有大规模非结构化数据的 Hadoop 集群。

还将会讨论一些 NLP 和文本挖掘领域中常见的用例，在这过程中，也会给出一些代码片段，以便帮助你完成相关的工作。具体来看三个会涉及绝大多数文本挖掘问题的主要示例。这些示例会告诉你如何通过大规模地执行 NLTK 来完成本书最初几章中所介绍的那些 NLP 任务。此外，还将通过几个例子来介绍如何在大数据条件下执行文本分类任务。

当然，机器学习和 NLP 还有另一高度规模化应用的问题就是它们是否可并行化。这里将会简单地讨论一下上一章中的一些问题，看看这些问题是否属于大数据问题，或者是否在某些条件下可以用大数据的方式来解决这些问题。

由于到目前为止所学习的大多数库都是用 Python 编写的，所以如何用 Python（Hadoop）来处理大数据也是本章的主要问题之一。

在阅读完本章之后，我们希望读者掌握以下内容。

- 能很好地了解 Hadoop、Hive 这些与大数据相关的技术，并在其条件下使用 Python。
- 根据教程一步一步地掌握如何在大数据条件下使用 NLTK、Scikit 和 PySpark。

10.1 在 Hadoop 上使用 Python 的不同方式

在 Hadoop 上运行一个 Python 进程的方式有很多种。在这里，将会讨论其中一些当前

最为流行的方式，并通过这些方式在 Hadoop 上用 Python 来实现流式的 MapReduce 作业[1]、Hive 中的 Python UDF 以及 Python hadoop 包装器。

10.1.1 Python 的流操作

通常，一个典型的 Hadoop 作业必须要被写成 map+reduce 函数的形式。用户需要根据给定任务来编写相应的 map+reduce 函数的实现。这些 mapper 和 reducer 通常是用 Java 来实现的。而与此同时 Hadoop 也提供了流操作的接口，用户可以基于这些接口来写一个 Python 封装器，并用其他任意一种语言来编写之前由 Java 所实现的 mapper 和 reducer 函数。接下来看一个用 Python 编写的单词计数示例。而且本章稍后还会介绍如何用 NLTK 库再实现一次。

> **提示：**
> 如果你还不太了解情况，可以看看下面链接中的资料:
> http://www.michael-noll.com/tutorials/writingan-hadoop-mapreduce-program-in-python/了解一下 Python 环境下的 MapReduce 模式。

10.1.2 Hive/Pig 下的 UDF

另一种使用 Python 处理大数据的方式就是在 Hive/Pig 中编写 **UDF（User Defined Function）**。这种方法的思路认为，在 NLTK 中所执行的大多数操作都是高度可并行化的。如说词性标注、标识化处理、词形还原、停用词移除以及 NER 这些都是可高度分布式执行的操作。因为其中的每一行内容都独立于其他行，所以在执行这些操作时不需要根据任何上下文。

因此，如果在集群上的每个节点上都部署了 NLTK 及其他 Python 库，也可以用 Python 来编写一些**用户定义函数（UDF）**，以借助 NLTK 和 scikit 这些库的功能。这是引用 NLTK 最简单的一种方法，对于 scikit 的大规模引用则更是如此。本章后续内容中具体介绍这两个库的情况。

10.1.3 流封装器

可以将不同组织所实现的各种封装器列成一份长长的列表，以便能让人明白 Python 在集群上可以执行的各项任务。这其中有一些封装使用起来其实相当简单，但问题是它们都有性能较差这个问题。我在下面也列出了一些，如果你想了解它们，可以去这些项目的网站阅读一下相关介绍。

---

① 译者注：MapReduce 是一种编程方式，主要用于针对大规模数据集的并行计算。

- Hadoopy。
- Pydoop。
- Dumbo。
- mrjob。

> **提示：**
> 如果想查看一份更详细的目前 Hadoop 上可供选择的 Python 库列表，读者可以参阅下面这篇文章：http://blog.cloudera.com/blog/2013/01/a-guide-topython-frameworks-for-hadoop/。

10.2 Hadoop 上的 NLTK

之前已经从一个库的角度对 NLTK 进行了充分的讨论，并介绍了它的一些最常用的函数。目前，NLTK 已经可以解决许多 NLP 问题，这其中有许多都是高度可并行化的方案。这也是为什么要试着在 Hadoop 上使用 NLTK 的原因。

在 Hadoop 上，运行 NLTK 的最佳途径就是将其安装在集群的所有节点上。这实现起来并不困难。有几种方式都可以做到这一点，例如可以将资源文件以流参数的形式发送。但通常都选择下面的第一个选项。

10.2.1 用户定义函数（UDF）

在 Hadoop 上运行 NLTK 的方法有很多，下面讨论一个实例，来看看它是如何用 NLTK 来实现一个 Hive UDF，以便完成并行式的标识化处理。

这个用例的操作可以分成以下步骤。

1．这里选择了一个只包含两列数据的小型数据集，然后在 Hive 中创建与之相对于的相同数据模式[1]：

ID	内容
UA0001	"I tried calling you. The service was not up to the mark"
UA0002	"Can you please update my phone no"
UA0003	"Really bad experience"
UA0004	"I am looking for an iPhone"

① 译者注：由于原文针对的是英文环境，所以这里的例句就不翻译了，下同。

2．现在，在 Hive 中创建相同的数据模式，该操作的 Hive 脚本如下：

Hive script

```
CREATE TABLE $InputTableName (
ID String,
Content String
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';
```

3．数据模式建构完成之后，下一步要做的就是针对每一个独立列中的内容进行类似于标识化的处理。所以，现在就会想在$ outTable 中设置另一个具有相同模式的列，并为其添加相应的标识列：

Hive script

```
CREATE TABLE $OutTableName (
ID String,
Content String,
Tokens String
)
```

4．现在模式方面的准备已经完成了，接下来要用 Python 来编写 UDF，以逐行读取上面的数据表，并对其执行 tokenize()方法。这一过程与第 3 章中所做的事情非常类似。这是一个与第 3 章中所有示例都很类似的函数。而且，如果你现在还想获得其 POS 标签、词形还原以及某 HTML 标记的话，只需要相应地修改一下这个 UDF 即可。下面就来看看 UDF 是如何查找相关标识的：

```
>>> import sys
>>> import datetime
>>> import pickle
>>> import nltk
>>> nltk.download('punkt')
>>> for line in sys.stdin:
>>>     line = line.strip()
>>>     print>>sys.stderr, line
>>>     id, content= line.split('\t')
>>>     print>>sys.stderr,tok.tokenize(content)
>>>     tokens =nltk.word_tokenize(concat_all_text)
>>>     print '\t'.join([id,content,tokens])
```

5．接下来，命名一下这个 UDF，如 nltk_scoring.py。

6．现在，用 TRANSFORM 函数来执行 Hive 的 insert 查询，以便将上面的 UDF 应用

到给定的内容中，将新列标记化并转储其标识：

Hive script

```
add FILE nltk_scoring.py;
add FILE english.pickle; #Adding file to DistributedCache
INSERT OVERWRITE TABLE $OutTableName
SELECT
        TRANSFORM (id, content)
    USING 'PYTHONPATH nltk_scoring.py'
    AS (id string, content string, tokens string )
FROM $InputTablename;
```

7. 如果你在上述过程中遇到了下面这样的错误信息，就说明 NLTK 机器数据组件没有被正确安装：

```
raiseLookupError(resource_not_found)
LookupError:
******************************************************************
****
  Resource u'tokenizers/punkt/english.pickle' not found. Please
  use the NLTK Downloader to obtain the resource: >>>
  nltk.download()
  Searched in:
    - '/home/nltk_data'
    - '/usr/share/nltk_data'
    - '/usr/local/share/nltk_data'
    - '/usr/lib/nltk_data'
    - '/usr/local/lib/nltk_data'
```

8. 如果该 Hive 作业能够被成功运行，就会得到一张名为 OutTableName 的表，具体如下：

ID	内容	
UA0001	"I tried calling you, The service was not up to the mark"	[" I", " tried", "calling", "you", "The", "service" "was", "not", "up", "to", "the", "mark"]
UA0002	"Can you please update my phone no"	["Can", "you", "please" "update", " my", "phone" "no"]
UA0003	"Really bad experience"	["Really"," bad" "experience"]
UA0004	"I am looking for an iphone"	["I", "am", "looking", "for", "an", "iPhone"]

10.2.2 Python 的流操作

下面来试试第二个选项：Python 流。根据现有的 Hadoop 流来编写自己的 mapper 和

reducer 函数，然后使用 mapper.py 来操作 Python 流，它的用法和 Hive UDF 非常类似。下面，就用同一个例子来示范一下 map-reduce 函数操作 Python 流的过程。这种方法提供了一种可以使用 Hive 表，甚至直接使用 HDFS 文件的选项。在这里，只示范如何读取相关的内容并将其标识化，并不会涉及任何 reduce 操作，但为了让学习有一定的完整性，我会在这里纳入一个虚拟的 reducer，该函数只负责转储相关结果。也正因为如此，这里可以完全忽略来自命令执行环境中的 reducer。下面是 Mapper.py 的代码：

Mapper.py

```
>>>import sys
>>>import pickle
>>>import nltk
>>>for line in sys.stdin:
>>>    line  = line.strip()
>>>    id, content = line.split('\t')
>>>    tokens =nltk.word_tokenize(concat_all_text)
>>>    print '\t'.join([id,content,topics])
```

接下来是 Reducer.py 的代码：

Reducer.py

```
>>>import sys
>>>import pickle
>>>import nltk
>>>for line in sys.stdin:
>>>    line = line.strip()
>>>    id, content,tokens = line.split('\t')
>>>    print '\t'.join([id,content,tokens])
```

下面再来看看 Hadoop 中执行 Python 流操作的命令：

```
hadoop jar <path>/hadoop-streaming.jar \
-D mapred.reduce.tasks=1 -file <path>/mapper.py \
-mapper <path>/mapper.py \
-file <path>/reducer.py \
-reducer <path>/reducer.py \
-input /hdfspath/infile \
-output outfile
```

10.3 Hadoop 上的 Scikit-learn

机器学习是大数据领域中另一个非常重要的用例。在这方面 Hadoop、scikit-learn 这些

框架就显得更尤为重要了，因为这是大数据环境下对机器学习模型进行评分的最佳选项之一。由于大型机器学习本身就是当前业界最热门的话题之一，所以在 Hadoop 这样的大数据环境中做这些事就显得更为重要了。当前，机器学习模型主要有两个方面的问题：在大数据环境下建模，以及针对大量数据的建模和评分。

为了加深对上述概念的理解，这里也将采用之前那张表中的相同示例数据，这其中包含了一些客户的评论信息。现在假设要用一份重要的训练样本来构建一个文本分类模式，并利用在第 6 章中学到的知识在该数据上构建出朴素贝叶斯算法、SVM 或逻辑回归模型。然后在进行评分的时候，可能需要面对的是针对海量数据的评分，如说客户评论这样的信息。另一方面，靠 scikit-learn 库在大数据环境中建模是做不到的，所以这里会需要用到像 spark / Mahot 这样的工具。另外，还会像在使用 NLTK 时一样用到某种预训练模型，并执行相同的步骤评分方法，下一节中将会介绍其建模过程。特别在处理处理文本挖掘类问题时，使用预训练模型来执行评分是非常合适的。当然，需要将这里的两个主要对象（vectorizer 和 modelclassifier）存储成某种序列化的 pickle 对象。

> **提示：**
> 这里提到的 pickle 其实是一个 Python 模块，主要用于实现序列化，即通过该模块，可以将对象以二进制状态存储在磁盘上，并且可以通过二次加载来使用。
> https://docs.python.org/2/library/pickle.html。

下面，我们要用 scikit 在本地机器上构建离线模型，并准备好 pickle 对象。举例来说，如果这里重现的是第 6 章中朴素贝叶斯算法的那个例子，就必须要将其中的 vectorizer 和 clf 转储成 pickle 对象：

```
>>>vectorizer = TfidfVectorizer(sublinear_tf=True, min_df=in_min_df,
stop_words='english', ngram_range=(1,2), max_df=in_max_df)
>>>joblib.dump(vectorizer, "vectorizer.pkl", compress=3)
>>>clf = GaussianNB().fit(X_train,y_train)
>>>joblib.dump(clf, "classifier.pkl")
```

以下是创建输出表的步骤，该表中包含了所有客户评价的历史记录。

1．在 Hive 中创建一个与之前示例相同的模式。下面就用 Hive 脚本来执行这一操作。这个输出表可能很巨大，具体到眼下这个例子，假设这其中包含了公司收到过的所有客户评论：

Hive script

```
CREATE TABLE $InputTableName (
ID String,
```

```
Content String
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';
```

2．构建组成输出表的各列，例如评分可信度等：

Hive script

```
CREATE TABLE $OutTableName (
ID String,
Content String,
predict String,
predict_score double
)
```

3．接下来，要用 Hive 中的 addFILE 命令将这些 pickle 对象加载到其分布式缓存中：

```
add FILE vectorizer.pkl;
add FILE classifier.pkl;
```

4．下一步就是编写 Hive UDF 了，会在这些函数中定义如何加载这些 pickle 对象。现在，这些对象的行为必须要先与其在本地的行为保持一致。在设置好这些 classifier 和 vectorizer 对象之后，接下来就可以使用测试样本了。在这里，测试样本只是一个字符串，然后生成输出的是 TFIDF 向量。vectorizer 对象现在属于预测类，可以当作概率类来使用：

Classification.py

```
>>>import sys
>>>import pickle
>>>import sklearn
>>>from sklearn.externals import joblib

>>>clf = joblib.load('classifier.pkl')
>>>vectorizer = joblib.load('vectorizer.pkl')

>>>for line in sys.stdin:
>>>    line = line.strip()
>>>    id, content= line.split('\t')
>>>    X_test = vectorizer.transform([str(content)])

>>>    prob = clf.predict_proba(X_test)
>>>    pred = clf.predict(X_test)
>>>    prob_score =prob[:,1]
>>>    print '\t'.join([id, content,pred,prob_score])
```

5．在编写完 classification.py 中的 UDF 之后，还必须要将这个 UDF 添加到分布式缓存中，然后有效地在输出表的每一行上将这个 UDF 以 TRANSFORM 函数的形式运行。其 Hive 脚本如下：

Hive script

```
add FILE classification.py;

INSERT OVERWRITE TABLE $OutTableName
SELECT
    TRANSFORM (id, content)
    USING 'python2.7 classification.py'
    AS (id string, scorestringscore string )
FROM $Tablename;
```

6．如果一切顺利，就会得到符合以下输出模式的输出表：

ID	内容	预测	评分可信度
UA0001	"I tried calling you, The service was not up to the mark"	投诉	0.98
UA0002	"Can you please update my phone no "	非投诉	0.23
UA0003	"Really bad experience"	投诉	0.97
UA0004	"I am looking for an iPhone "	非投诉	0.01

如你所见，输出表中包含了过去所有的客户评论记录，以及这些评论是否属于投诉的预测和预测的可信度。当然例子采用的是 Hive UDF，但类似的过程也可以用 Pig 和 Python 流的方式来做，其做法与用 NLTK 非常类似。

这个例子所演示的是如何在 Hive 上对机器学习模型进行评分。在下一个例子中，我们将要讨论如何在大数据环境中构建机器学习/NLP 的模型。

10.4 PySpark

先来回顾一下之前是如何在 Hadoop 上构建机器学习/NLP 模型，以及如何对这个 Hadoop 上的 ML 模型进行评分的。在上一节中，我们较为深入地讨论了一下评分的第二个选项。与对规模较小的数据集进行抽样不同的是，我们这次要对规模较大的数据集进行评分，并使用 PySpark 库来逐步构建大型的机器学习模型。当然，我们在这里会再次使用到与之前相同模式的相同的运行数据：

ID	评论	类别
UA0001	I tried calling you, The service was not up to the mark	1
UA0002	Can you please update my phone no	0
UA0003	Really bad experience	1
UA0004	I am looking for an iPhone	0
UA0005	Can somebody help me with my password	1
UA0006	Thanks for considering my request for	0

通常，我们要考虑的是近 10 年来评价的组织模式。但现在要改用一个小型的样本，先为其创建分类模型，然后再用预训练模型对所有评论进行评分。下面，就通过一个例子来逐步演示一下这个过程，具体说明一下如何用 PySpark 库来构建文本分类模型。

首先来导入一些模块。从 SparkContext 开始，这更多的是一个配置性的模块，在这里可以通过更多的参数来提供信息，如应用程序的名称等。

```
>>> from pyspark import SparkContext
>>> sc = SparkContext(appName="comment_classifcation")
```

提示：
如果想了解关于 PySpark 库的更多信息，你可以阅读一下下面这篇文章：
http://spark.apache.org/docs/0.7.3/api/pyspark/pyspark.context.SparkContext-class.html。

接下来要读取这个由制表符分隔的文本文件。要将该文件读取到 HDFS 上，文件的体积可能会很大（～Tb/Pb）：

```
>>> lines = sc.textFile("testcomments.txt")
```

现在，lines 就成为了包含语料库中所有行的一个列表：

```
>>> parts = lines.map(lambda l: l.split("\t"))
>>> corpus = parts.map(lambda row: Row(id=row[0], comment=row[1],
class=row[2]))
```

这部分代码的作用获取是字段列表，其依据是各行中分隔字段的“\ t”符。

下面要将不同 RDD 对象中的语料库分解成[ID, comment, class (0,1)]的形式：

```
>>> comment = corpus.map(lambda row: " " + row.comment)
>>> class_var = corpus.map(lambda row:row.class)
```

一旦收集完了评论信息，接下来的处理过程就与第 6 章中所做的事非常类似了。这里会用 scikit 库来执行标识化处理、散列化 vectorizer 以及用 vectorizer 来计算 TF、IDF 和 tf-idf。

下面来看看如何创建标识化处理、词汇频率和反向文档频率，其代码如下：

```
>>> from pyspark.mllib.feature import HashingTF
>>> from pyspark.mllib.feature import IDF
# https://spark.apache.org/docs/1.2.0/mllib-feature-extraction.html

>>> comment_tokenized = comment.map(lambda line: line.strip().split(" "))
>>> hashingTF = HashingTF(1000) # to select only 1000 features
>>> comment_tf = hashingTF.transform(comment_tokenized)
>>> comment_idf = IDF().fit(comment_tf)
>>> comment_tfidf = comment_idf.transform(comment_tf)
```

接下来，要将该类与 tfidf RDD 对象进行合并：

```
>>> finaldata = class_var.zip(comment_tfidf)
```

然后来做一个典型测试，进行训练取样：

```
>>> train, test = finaldata.randomSplit([0.8, 0.2], seed=0)
```

现在来执行主分类命令，该命令与 scikit 库非常相似。这里使用的是逻辑回归算法，这是一种使用度很广的分类器。pyspark.mllib 中提供了各种可用的算法。

> **提示：**
> 如果想了解更多关于 pyspark.mllib 的信息，可以参考以下链接中的资料：https://spark. apache.org/docs/latest/api/python/pyspark.mllib.html。

下面是朴素贝叶斯分类器的示例：

```
>>>from pyspark.mllib.regression import LabeledPoint
>>>from pyspark.mllib.classification import NaiveBayes
>>>train_rdd = train.map(lambda t: LabeledPoint(t[0], t[1]))
>>>test_rdd = test.map(lambda t: LabeledPoint(t[0], t[1]))
>>>nb = NaiveBayes.train(train_rdd,lambda = 1.0)
```

```
>>>nb_output = test_rdd.map(lambda point: (NB.predict(point.features),
point.label))
>>>print nb_output
```

nb_output 命令中包含了对于测试样本的最终预测。这里需要了解的最重要的一件事是，上面的样本只有不到 50 行的内容，而在代码中所构建的是一个符合行业标准的文本分类操作，可以处理 PB 级规模的训练样本。

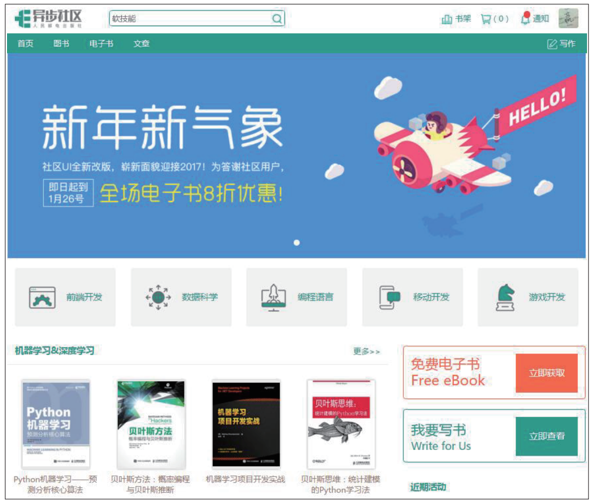
10.5 小结

下面来对本章做个小结，本章的目标是介绍如何在大数据环境中使用目前所学到这些概念。在这里，具体介绍了如何在 Hadoop 中使用 NLTK 和 scikit 这些库。也讨论了如何对机器学习模型或基于 NLP 的操作进行评分。

为了介绍这些内容，本章主要列举了 3 个业界最常见的用例来做演示。只要理解了这些示例，我们就能用好 NLTK、scikit 和 PySpark 这 3 个库中的大部分函数了。

本章基本上算是对大数据环境下的 NLP 和文本挖掘所做的一个速食性的简介。这毕竟是当前最为热门的话题之一，示例代码中所讨论到的每个术语和工具本身都足以写成一本书。这里试图给读者提供某种黑客方法，以便对大数据这种规模下的文本挖掘问题做一个介绍。我希望并鼓励读者多阅读一些和大数据有关的技术资料，如 Hadoop、Hive、Pig 和 Spark 等，并自己去试着实现一下本章所介绍的这些示例。

欢迎来到异步社区！



异步社区的来历

异步社区（www.epubit.com.cn）是人民邮电出版社旗下 IT 专业图书旗舰社区，于 2015 年 8 月上线运营。

异步社区依托于人民邮电出版社 20 余年的 IT 专业优质出版资源和编辑策划团队，打造传统出版与电子出版和自出版结合、纸质书与电子书结合、传统印刷与 POD 按需印刷结合的出版平台，提供最新技术资讯，为作者和读者打造交流互动的平台。

社区里都有什么？

购买图书

我们出版的图书涵盖主流 IT 技术，在编程语言、Web 技术、数据科学等领域有众多经典畅销图书。社区现已上线图书 1000 余种，电子书 400 多种，部分新书实现纸书、电子书同步出版。我们还会定期发布新书书讯。

下载资源

社区内提供随书附赠的资源，如书中的案例或程序源代码。

另外，社区还提供了大量的免费电子书，只要注册成为社区用户就可以免费下载。

与作译者互动

很多图书的作译者已经入驻社区，您可以关注他们，咨询技术问题；可以阅读不断更新的技术文章，听作译者和编辑畅聊好书背后有趣的故事；还可以参与社区的作者访谈栏目，向您关注的作者提出采访题目。

灵活优惠的购书

您可以方便地下单购买纸质图书或电子图书，纸质图书直接从人民邮电出版社书库发货，电子书提供多种阅读格式。

对于重磅新书，社区提供预售和新书首发服务，用户可以第一时间买到心仪的新书。

用户账户中的积分可以用于购书优惠。100 积分 =1 元，购买图书时，在 

里填入可使用的积分数值，即可扣减相应金额。

特 别 优 惠

购买本书的读者专享**异步社区购书优惠券**。

使用方法：注册成为社区用户，在下单购书时输入 57AWG 使用优惠码，然后点击“使用优惠码”，即可享受电子书8折优惠（本优惠券只可使用一次）。

纸电图书组合购买

社区独家提供纸质图书和电子书组合购买方式，价格优惠，一次购买，多种阅读选择。



社区里还可以做什么？

提交勘误

您可以在图书页面下方提交勘误，每条勘误被确认后可以获得100积分。热心勘误的读者还有机会参与书稿的审校和翻译工作。

写作

社区提供基于 Markdown 的写作环境，喜欢写作的您可以在此一试身手，在社区里分享您的技术心得和读书体会，更可以体验自出版的乐趣，轻松实现出版的梦想。

如果成为社区认证作译者，还可以享受异步社区提供的作者专享特色服务。

会议活动早知道

您可以掌握 IT 圈的技术会议资讯，更有机会免费获赠大会门票。

加入异步

扫描任意二维码都能找到我们：

异步社区	微信服务号	微信订阅号	官方微博	QQ 群：436746675

社区网址：www.epubit.com.cn

官方微信：异步社区

官方微博：@ 人邮异步社区，@ 人民邮电出版社 - 信息技术分社

投稿 & 咨询：contact@epubit.com.cn

NLTK基础教程

用NLTK和Python库构建机器学习应用



自然语言处理（NLP）属于人工智能与计算机语言学的交叉领域，处理的是计算机与人类语言之间的交互问题。随着人机交互需求的日益增长，计算机具备处理当前主要自然语言的能力已经成为了一个必然趋势。NLTK正是这一领域中一个强大而稳健的工具包。

在这本书中，我们首先会介绍一些与NLP相关的知识。然后，我们会探讨一些与数据科学相关的任务，通过这些任务来学习如何从零开始构建自定义的标识器和解析器。在此过程中，我们将会深度探索NLP领域的基本概念，为这一领域各种开源的Python工具和库提供具有实践意义的见解。接下来，我们将会介绍如何分析社交媒体网站，发现热门话题，进行舆情分析。最后，我们还会介绍一些用于处理大规模文本的工具。

在阅读完本书之后，您将会对NLP与数据科学领域中的概念有一个充分的了解，并能将这些知识应用到日常工作中。

如果您是NLP或机器学习相关领域的爱好者，并有一些文本处理的经验，那么本书就是为你量身定做的。此外，这本书也是专业Python程序员快速学习NLTK库的理想选择。

通过本书，你将学会：

- 了解自然语言的复杂性以及机器对它们的处理方式。
- 如何利用标识化处理手段清理文本歧义，并利用分块操作更好地处理数据。
- 探索不同标签类型的作用，并学习如何将句子标签化。
- 如何根据自己的需要来创建自定义的解析器和标识器。
- 如何构建出具有拼写检查、搜索、机器翻译以及问答系统等功能的实用程序。
- 如何通过信息爬取与捕获的手段对相关数据内容进行检索。
- 如何通过特性的提取与选取，构建出针对不同文本的分类系统。
- 如何使用各种第三方Python库，如pandas、scikit-learn、matplotlib、gensim。
- 如何对社交媒体网站进行分析，包括发掘热门话题、舆情分析等。

异步社区
人民邮电出版社
www.epubit.com.cn

异步社区 www.epubit.com.cn
新浪微博 @人邮异步社区
投稿/反馈邮箱 contact@epubit.com.cn

ISBN 978-7-115-45257-3

美术编辑：董志桢

分类建议：计算机／机器学习／自然语言处理
人民邮电出版社网址：www.ptpress.com.cn